# ULB
# INFO-F403 - Introduction to language theory and compiling
# Introduction to language theory and compiling

Bastogne Jérôme,
Hereman Nicolas

Academic year 2015-2016 - October 27, 2015

# Chapter 1

# Part 1 - Scanner

## 1.1 Lexical analyser using Jflex

To distinguish between comments and the grammar of the language or lexical units we are using two exclusive states. The initial one <YYINITIAL> and the comment state <COMMENT_STATE>. While being in the initial state, if we find the comment Regular Expression, we just switch from state, otherwise we analyse the current token. While being in the comment state, we just wait for the second comment Regular Expression to switch back to the initial state ignoring every other token. In the initial state we throw an exception if we found an unknow token.

## 1.2 Regular Expressions

```
VarName     =     [A–Za–z][A–Za–z0–9]*
Number      =     [0–9]+
Comment     =     "co"("\n" | " ")
```

We can see here our used regex for the regular expressions.
A [Number] represents a numerical constant, and is made up of a string of digits only. The + indicates there has to be at least one digit to be a number.
A [VarName] identifies a variables, which is a string of digits and letters, starting with a letter (this is case sensitive). That is exactly what our regex does, at least one letter followed by any number of digits and letters.
A [Comment] starts and ends with a co keywords. Our comment regex is made like this to avoid VarNames starting by "co" to be considered as comments.
We assumed here that voluntary errors that appear in the code we want to compile would be handled later in the project, while or after parsing. We think it isn't the lexical analyzer's job. Because for now, if the lexical analyzer finds the token "9azer", wich is not allowed by the compiler for SUPRALGOL, he will consider two separate lexical units : "9" and "azer". We think this should be handled and reported in the next steps of the compiler process.

All the others regex are simply plain text keyword like "read", ";" "while", ...

## 1.3 Identifiers

Each time we find a variance with jflex we check if it is the first time we encounter it. If it is we had it in the identifiers map with the variance as a key and the line as a value. We use a TreeMap so the key are stored in an alphabetical order.

## 1.4 Test files

We tried to make some tests aiming for a spread use of the grammar of the language and some lexical units.