# INFO-F-403: Project Super-Fortran Part 1 Lexer analyser

Sefu Kevin and Abou Zaidi Ahmed

October 21, 2018

## 1 Introduction

In this project we were requested to design and write a compiler for Super-Fortran. It is a very simple imperative language. The project is divided in 3 parts and for this first part we had to produce the lexical analyser of the compiler.

The lexical analyser has been generated using JFlex which is a tool that take a specification (.flex file) as input with a set of regular expression and with each one we can associate java code that will be executed in each match.

## 2 Regular expressions

**`Upper = [A-Z]`** Here we define a macro of a range that match any upper letter from A to Z.

**`Lower = [a-z]`** This is the same as the previous one but for lowers letters.

**`Alpha = {Upper}|{Lower}`** The language of alpha define the union of the languages of the 2 previous macros.

**`Numeric = [0-9]`** The range of all decimals digits that will be useful for matching numbers.

**`AlphaNum = {Alpha}|{Numeric}`** A macro that matches all alphanumerics characters.

`NZero = [1-9]` We need non zero digit macro because a number literal can't begin with 0.

`Zero = "0"` We need a macro for matching single zero which is the only number that begins with 0.

`VarName = {Lower}({Numeric}|{Lower})*` VarName is the macro defining a variable which starts with a lower case characters and contains only digits and lower case characters.

`EndLine = "\n"` This macro exist because it will be repeated several time and we want to avoid magic constants.

`Number = {Zero}|({NZero}{Numeric}*)` A number is a zero or a non zero digit followed by an arbitrary number of digits.

`ProgName = {Upper}{AlphaNum}*({Lower}|{Numeric}){AlphaNum}*` A program name start with an upper case characters and is said to be a non upper case word. The choice of the meaning of a non upper case word will be explained in the next section.

`BeginComment = "/*"` The begin of a multi line comment.

`Ignore = " " | "\t"` The ignored characters by the lexer.

## 3 Explanation of choices and hypotheses

### 3.1 Macro definitions

We decided to use macros everywhere except when we face one character or a concatenation of two characters that will not be reused in the flex file because macros make the code more readable and the changes in the input language easier.

### 3.2 Unexpected tokens

Unexpected tokens are words that do not match with any regular expression. We consider that they should not be simply ignored by the lexical analyser that's why an exception is thrown and for this reason we defined a macro `Ignore` for characters that should be ignored and are not a part of the grammar of super Fortran.

### 3.3 Non upper case word

For the matching of the Progname we had two choices on how to interpret a non upper case word. First we can define it as a word that contains at least one lower case character or the other option is a word that contains at least one digit or one lower case character.

We took the second one because a word that contains a digit can't be confused with a keyword.

## 4 Bonus question

The difficulty with nested comments problem was to find a regular expression that recognize them. In fact the nested comments problem is similar to the problem of opening/closing parenthesis which is not a regular language. For this reason the solution we found was to use jflex states in which we can put java code.

Nested comment requires to count (or to use a stack), opening and closing strings for comment. With a java integer as a counter that increments when we find an opening sequence and decrements on a closing sequence then we can track if we are currently inside a comment. It is the case when the counter is positive. When the counter is equal to 0 we are outside comments and when the counter is negative there is more closing sequences than needed and this should be considered as a syntax error.

Moreover we know that $L_i$ is a regular language iff it exists a finite automata A such that $L(A) = L_i$ by Kleene's theorem, let's call $L_{nc}$ the language of nested comments and the word $w$ with $n$ opening/closing comment belongs to the language and should be accepted by the automata. By running A on $w$ there will be at least one state that will be repeated by pigeon's hole principle. It will be repeated by Kleene's closure or by a loop between states. Hence this part can be repeated an arbitrary amount of time and one additional repeating can take the word out of $L_{nc}$. Meaning that $L(A) \neq L_{nc}$.

We did not implement nested comment support because they are forbidden and useless.

## 5 Tests files desciption

**`01-Read.sf` and `00-Factorial.sf`** Those two files are provided by the teacher assistant.

**`02-Factorial-nested-comment.sf`** This file contains nested comments to ensure that an error is raised.

**`03-Fibonacci.sf`** This file contains an implementation of Fibonacci number.

**`04-Factorial-unexpectedword.sf`** This file contains an upper case word that is not a keyword. An unexpected token will be raised on reading.

## 6 Conclusion

As we said in the introduction we implemented a lexer for the Super-Fortran programming language.

During this project we learned how to build a lexical analyser. It allowed us to learn more about the first step of a compiler and to understand deeply how it works. We also learned that we can build a lexer without writing a tokenization algorithm.