INFO-F403 Project part 1 : Lexical analyser

Vincent Gailly 547819 - Oruç Kaplan 540662

October 2022

1 Introduction

This report contains a more detailed explanation of our project and the implementations made. We will also explain the tests that have been done and we will propose a solution to the nested comments in Fortress.

2 Regular expressions

2.1 VarName

A VarName can only be composed of lower case letters and digits which are identified by the following regular expressions:

- **AlphaLowerCase** $= [a - z]$
- **Numeric** $= [0 - 9]$

This one must absolutely start with a lower case letter. So we start by putting a lower case letter in the regex before concatenating it with either a digit or a lower case letter:

{AlphaLowerCase}({AlphaLowerCase}|{Numeric})

As the VarName can be followed by any number of lowercase letters or digits we need to add a ***** which will indicate that the first lowercase letter can be followed by 0 or several times the ({AlphaLowerCase}|{Numeric}) set:

{AlphaLowerCase}({AlphaLowerCase}|{Numeric})*

2.2 ProgName

A ProgName must start with an uppercase letter and then can be followed by any alpha numeric character. That is, lower or upper case letters as well as numbers. But in no case can it be completely capitalized. To do so, we use then new following regular expressions:

- **AlphaUpperCase** $= [A - Z]$
- **Alpha** $= \{AlphaUpperCase\}|\{AlphaLowerCase\}$
- **AlphaNumeric** $= \{Alpha\}|\{Numeric\}$

We end up with this regular expression:

{AlphaUpperCase}{AlphaNumeric}*{AlphaLowerCase}+{AlphaNumeric}*

The regex start with {AlphaUpperCase}, that force to have at least one uppercase letter at the beginning. Then we can have anything from uppercase, lowercase letters and numbers. But we need at least one lowercase letter in any position after the first letter of the regex. That's why, there is a {AlphaLowerCase}+ which force to have at least one lowercase letter surrounded by {AlphaNumeric}*.

2.3 Number

A Number is a string of digits so it can only be composed of Numeric. But we do not allow a 0 to lead, so the first number will be in : $[1-9]$.

To create all the number possible we need to concatenate $[1-9]$ with {Numeric}*, so we have :

[1-9]{Numeric}*

The problem with this previous regex is that it does not allow a simple zero, so we need to change it like this :

([1-9]{Numeric}*)|0

2.4 Short Comment

A short comment is all the symbols appearing after :: and up to the end of the line. To do this we need to add a new regular expression :

- **LineTerminator** = $\backslash r \mid \backslash n \mid \backslash r \backslash n$

Now we need to first identify a :: which is follow by whathever except a LineTerminator follow by a LineTerminator, which give us :

::[^\r\n]*{LineTerminator}?

2.5 Long Comment

A long comment is any sequence of symbols appearing between two %%. That means, the long comment will start with a %% and then will end up with %%. A long comment can also continue on multiple lines. At first we used a regular expression which was the following :

"%%" ~ "%%" Which is equivalent to **"%%" [^"%%"]* "%%"**

In this regular expression appears the "~" (tilted). It takes the form of "~expr" and allows to match everything up to (and including) the first occurrence of a text corresponding to the expression "expr".

But finally we decided to go through "COMMENT" and "YYINITIAL" states, the first one indicating that we are in a comment. In addition to that, we added an integer variable that we incremented each time we crossed a "%%" whose appearance number was even and that we decremented when it was odd. This modification allowed us to throw an exception in the %eof of JFLEX when a long comment was not closed before the end of the file (by looking if the variable is equal to 0 or not).

3 Example files

We created several files in the test folder. They allow us to test our lexical analyser, in particular the test of all the regular expressions and symbols, if the variables are sorted correctly and all the strange and ambiguous cases that the comments can generated.

3.1 All symbols and regular expressions

AllRegex.fs

In this file all the symbols and regular expression are tested. That means all the classical tokens like : "BEGIN", "END", "IF", "+", "-", ":="... And also the different regex with special cases for variable names, program names and number. For example, testing that a program name must start with an upper case letter and not being entirely capitalized etc...

3.2 Variables sorting

SortVariable.fs

In this file, we test if all the variables are sorted alphabetically at the end of the output during the variables listing.

3.3 Comments

Comments.fs

In this file, we check the different possible scenarios with the comments. Comments which contains other tokens, classic comments, empty comments, multi line comments and ambiguous cases with comments followed by code, then other comments in between.

3.4 Factorial

Factorial.fs

This file is a classical Fortress code provided on the UV, which allow us to test if everything works properly.

4 Bonus

The difficulty is that the characters used to start a long comment are the same as those used to end it. It is therefore impossible to determine only with the information we currently have whether the comment block ends or whether it is a nested comment that begins...

A simple solution would be to differentiate the characters to start a comment from the characters to end it. We could then have a stack that would increment when we meet a symbol for the beginning of a comment and that would decrement with a symbol for the end.

Here is an example of a solution if we use the *%option stack* of Jflex and transform the symbol at the beginning of a long comment into /% and at the end into %/ :

```
%x comment
%%
"/%" {
  yy_push_state(comment);
}
<comment>{
  "%/" {
    yy_pop_state();
  }

  "/%" {
    yy_push_state(comment);
  }
}
%%
```