Compiler Project | 02 SCANNER

By: Mónica AYALA, Davide AVESANI, Théodore PRÉVOT

Introduction

The objective of this second part of the project is to develop a scanner by using FLEX, a programming tool based on LEX.
This scanner takes in input some text written in a specific language and returns as output the set of tokens defined by the language.
Our objective is therefore to define a set of rules for the FLEX program based on the grammar we have defined on the previous step of the project.

Installation and compilation

To quickly start with the implementation of the analyzer, we decided to install FLEX through sudo apt, by using the following command on linux:
$*sudo apt-get install flex*
After successfully installing the program, we started working from the content present on the VM, in particular by editing the scanner.l file as suggested on the delivery.
To compile the file run this command:
$*make re*
If the project is compiled successfully the resulted output should be

To execute the code run:
$./out/compo tests/basic.md
The test input file can be seen and modified inside the file:
/tests/basic.md

Define the rules to match grammar tokens

To define the rules we first started by looking at lexer documentation on the internet and also the given example.
We then defined our grammars with aliases to easily reuse matching expressions. Once all our tokens were defined we used modes in order to detect the *beginning* of a rule and then we could pop the last mode from the mode stack at the end of the same rule. This allowed us to detect multiple rules. For example ** test -- strike-- ** would be a bold rule with a strikethrough rule *inside*. With the mode stack it was quite easy to realize all the matching rules. When detecting the beginning or the ending of each rules we printed the token using the provided function “print_token(const char*)”;

External Tools

To check our regex easily, we used the tool https://regex101.com/ to easily test our regex used in flex.
With this tool we had real time feedback and explanation on regex and we could put any number of strings to check at the same time to ensure which one should match and which should not.

SVG rules

One of the most challenging parts of the project was to define the rules for SVG. In fact, this element contains various elements such as line, polyline, circle ecc, as shown in the following picture.

```
```xsvg:0,0,100,100
line 0,0 100,100 red
polyline 0,0 100,0 100,100 0,100 #06ac06
polygon 45,45 45,55 55,55 55,45 #ffd700 #000
circle 50,50 5 white white
ellipse 0,50 10 25 cyan yellow
rect 80,70 20 30 purple
text 50,50 "YAY" middle
```
```

The complete grammar for this element can be found in the grammar file.
To handle with it we started by defining the rule for entering the SVG mode which means matching the string
```*xsvg:*
Once done we defined all the rules for matching all the elements that can be found inside the SVG elements, and returning the specific tokens as defined in the grammar file.
We finally exit the SVG mode when encountering the string ```.
```