Introduction to Language Theory and Compilation: Exercises

Session 11: `lex/flex` scanner generator

Introduction

- `lex` is a tool that can automatically generate *scanners* (lexical analysers) from a *specification*.
- It's often used in conjunction with `yacc` (parser generator) which will be the subject of the next session.
- `JFlex` is a free Java implementation of `lex`, whereas `CUP` is a free Java implementation of `yacc`.

`lex` specifications

- The input of `lex` is a *specification* made of pairs of regular expressions and source code snippets.
- `lex` uses this information to generate source code that implements the corresponding scanner under the guise of a function called `yylex()`.
- The obtained executable analyses its own input, searches for occurrences of the specified regular expressions, and executes the corresponding source code.

Format of `lex` specifications

- A `lex` specification is broken up in three parts separated by a line with %% (can be switched following the implementation):
  1. arbitrary programming code to be prepended in the output scanner program
  2. regular expression definitions and arbitrary programming code (between `%{` and `%}`) to be inserted at the start of the scanner program
  3. translation rules (token identification by regular expression and associated source code pairs)
- The usual file extension for such a file is `.l` or `.lex`

Java – Example lex file

```
%{
/* Arbitrary Java code to be prepended to generated code */
import static java.Math.*;
%}

%class Lexer
%standalone
%unicode

number [0-9]
letter [a-zA-Z]
identifier {letter}({number}|{letter})*
integer ({number})+

%%

{identifier}  { System.out.println("ID: "+yytext()+" (length: "+yylength()); }
{integer} { System.out.print("Integer: "+new Integer(yytext()));}
```

Java – Compiling

To obtain the scanner executable :

1. Generate the scanner code with
   `java JFlex.Main myspec.flex`
   … which creates `Lexer.java` (%class option)
2. Compile the code generated by `JFex` into a class file:
   `javac Lexer.java`
   … which creates `Lexer.class`
3. Run it with `java Lexer <input file>`

You can find JFlex on `http://jflex.de`.

ULB

Exercise 1-2

1. Write a scanner that outputs its input file with line numbers in front of every line.
2. Write a scanner that outputs the number of alphanumeric characters, alphanumeric words and alphanumeric lines in the input file.

Lexer - States

Can use states in the lexer (**exclusive** or inclusive).
They must be declared with *%***x***state* State1,State2,... in the options list
The default state is **YYINITIAL** and a state can be used as:

$$\begin{aligned} < State\_Name > \quad & \{ \\ regex \quad & \{action\} \\ \ldots \\ \} \end{aligned}$$

Exercise 3-4

3. Write a scanner that only shows comments in the input file. Such comments are comprised within curly braces `{ }`.
4. Write a scanner that transforms the input text by replacing the word "compiler" with "ewww" if the line starts with an "a", with "???" if it starts with a "b" and by "profit!!!" if it starts with a "c".

Exercise 5

Write a *lexical analysis function* that recognises the following *tokens*:

- Decimal numbers in scientific notation (i.g. -0.4E-1)
- C99 variable identifiers (start by an alpha, followed by arbitrary number of alphanumeric or underscore)
- Relational operators ($<$, $>$, $==$, $!=$, $>=$, $<=$, $!$)
- The `if`, `then` and `else` keywords

The point of this function is then to be used by a `yacc` implementation, `cup` for `Java`.