Compilers

Spring 2024

Regular Expressions Exercises in ANTLR

Sample Exercises

Faculdade de Engenharia da Universidade do Porto
Departamento de Engenharia Informática

Problem 1

Develop a *lexer* using the ANTLR lexical token analyzer to detect valid IP specification in an input file. Recall, that IP addresses are expressed as a set of four numbers — an example address might be 192.158.1.38. Each number in the set can range from 0 to 255. So, the full IP addressing range goes from 0.0.0.0 to 255.255.255.255. You may choose to have a very permissive description of tokens that can detect sequences whose numeric values are not in the 0 to 255 range and then in a subsequent pass, determine which are in fact valid IP specifications.

Problem 2

The RISC-V instruction set architecture (or ISA) defines a series of assembly mnemonics for its instructions. For instance, `lw x10, 0(x2)` corresponds to the instruction that loads a register with a 32-bits word from memory and `addi x1, x0, 1` adds 1 to the value of register `x0` and stores the result in register `x1`. In addition, ISA usually define a specific range for the registers, in this specific case for x0 through x32 and for the immediate values (specified in integers) to be mapped to unsigned values in 12 bits. As an example, the -1 integer value will correspond to the 12-bit binary value of 111111111111 or in hexadecimal to 0xFFF.

You are now asked to develop a *lexer* using the ANTLR lexical token analyzer for the simple instructions of load word and store word, as well as simple register-only and register instructions with immediate values. Below you have valid and invalid examples of the instruction you are asked to detect.

```
lw x10, 0(x2)       OK
sw x10, 0(x2)       OK

lw x33, 0(x2)       Wrong. x33 invalid register
sw 0(x3), x2        Wrong. Invalid order. Register needs to proceed offset calc.

addi x1, x2, x0     Wrong. Last field must be an immediate value
add x1, x1, 1       Wrong. Last field must be a register id
```

You do not have to be exhaustive, and detecting some of the invalid and valid example is good enough for the purposes of this exercise.

Problem 3

In many contexts there is the desire to use mailing lists, where valid e-mails are separated by the comma character “,”. In this exercise you are asked to develop a *lexer* using the ANTLR lexical token analyzer to detect a list of valid e-mails, where are e-mail has the following structure. It needs to begin by a numeric or alpha-numeric character and contain only other characters such as “_” or the “-“. After the name in the e-mail, it needs to contain the “@” character followed by a sequence of dot-separated identifiers. The suffix of the e-mail needs to be either “org”, “com”, “gov” or one of the various country e-mail prefixes such as “pt” (for Portugal), “es” (for Spain) and so forth (you need not to verify all the countries…)

Problem 4

Develop a *lexer* using the ANTLR lexical token analyzer to detect a variable declaration in PASCAL. In PASCAL a variable is declared as shown below, where here for simplicity, we focus only on integer and real variables. The variable names are defined by the common definition of programming languages such as C, but here for simplicity, assume that a variable name can be a finite string of alphanumeric characters, obviously excluding the sequence “var”, and which must begin with a letter.

```
var a, b : integer ;
var x[10] : real ;
```

Validate your implementation for the selected set of examples above as well as for an example that does not comply with this structure, as is the case where there is a “var” string missing.

Problem 5

Assume that the input to your program is a CSV file (comma-separated value) and that between commas you have a pair of strings (separated by spaces or tab characters). Each of these pairs is a “up” number followed by a positive integer number (between 0 and 20). You should detect if the up number is valid by ensuring that you only accept “up” numbers with numeric prefixes that are either 2023 and 2024 and overall have a total of 9 numeric characters. Other numbers, while successfully scanned should be flagged as erroneous An example of such a string is shown below:

```
202207189 10, 202310789 19, 202404500 18, 20200001 10
```

You are asked to develop *lexer* in ANTLR and the corresponding processing code to detect, line-by-line all the valid and invalid pairs of “up” numbers and the corresponding integer values. The output of your program, which should read an entire CSV file, should be a list of the “up” numbers with a corresponding number greater or equal to a specified numeric argument to your program. For the example above, the specification of the value 18 as this numeric argument would yield the output of your program as where the first integer is the number of “up” numbers to follow.

```
2, 202310789, 202404500
```