Universität Stuttgart

Programming Paradigms Prof. Dr. Michael Pradel

Software Lab, University of Stuttgart, Winter 2019/2020

Exercise 1 Syntax: Regular Expressions, Grammars and Scanners

(Deadline for uploading solutions: Oct 27, 2019, 11:59pm Stuttgart time)

The materials provided for this homework are:

- a pdf file with the text of the homework (this);
- a zip file with **the folder structure and the templates that must be used for the submission**.

The folder structure is shown in Figure 1.

```
exercise1
├── task1
│   ├── syntaxA
│   │   └── answerA.txt
│   └── syntaxB
│       └── answerB.txt
├── task2
│   ├── syntaxC
│   │   ├── invalidC.txt
│   │   └── validC.txt
│   └── syntaxD
│       ├── invalidC.txt
│       └── validC.txt
├── task3
│   ├── command1
│   │   └── tree1.json
│   ├── command2
│   │   └── tree2.json
│   └── command3
│       └── tree3.json
└── task4.zip

10 directories, 10 files
```

Figure 1: Folder structure to be used for the solution.

The submission must be compressed in a zip file using the given folder structure. The name of folders and files must not be changed or moved, otherwise the homework will not be evaluated.

There are four tasks, which contribute a specific percentage to the overall points for this exercise.

Notes about the symbols used in this exercise:

- *blue tokens* are non-terminals;
- *black tokens* are terminals;
- $*$ is the Kleene star symbol;
- $*$ is a terminal symbol (e.g., multiplication symbol);
- | means "or";

1 Task I (10% of total points of the exercise)

You are given two sets of production rules. For each set, the question is: Do these rules describe a regular expression or a grammar?

1.1 Syntax A

$$
\begin{aligned}
&\textit{start} \longrightarrow \textit{sentence}\\
&\textit{sentence} \longrightarrow \textit{the qualified\_noun verb} \mid \textit{pronoun verb}\\
&\textit{qualified\_noun} \longrightarrow \textit{adjective noun}\\
&\textit{noun} \longrightarrow \textit{student} \mid \textit{programmer} \mid \textit{engineer}\\
&\textit{pronoun} \longrightarrow \textit{he} \mid \textit{she}\\
&\textit{verb} \longrightarrow \textit{talks} \mid \textit{studies} \mid \textit{works}\\
&\textit{adjective} \longrightarrow \textit{expert} \mid \textit{smart} \mid \textit{new}
\end{aligned}
$$

Write the following into the file *exercise1/task1/syntaxA/answerA.txt* as show in Figure 2: ***GRAMMAR*** if the above is a grammar or ***REGEX*** if it is a regular expression (in capital letters).

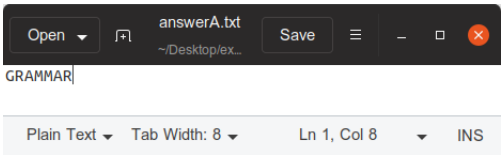

Figure 2: Example of the file containing the answer of task 1.

1.2 Syntax B

$$
\begin{aligned}
&\textit{start} \longrightarrow \textit{wishlist}\\
&\textit{wishlist} \longrightarrow \textit{product} \ \$ \ \textit{price} \mid \textit{product} \ \$ \ \textit{price} \ , \ \textit{wishlist}\\
&\textit{product} \longrightarrow \textit{car} \mid \textit{computer} \mid \textit{smartphone}\\
&\textit{price} \longrightarrow \textit{non\_zero\_digit digit}*\\
&\textit{non\_zero\_digit} \longrightarrow 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9\\
&\textit{digit} \longrightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9
\end{aligned}
$$

Write the following into the file *exercise1/task1/syntaxB/answerB.txt* as show in Figure 2: ***GRAMMAR*** if the above is a grammar or ***REGEX*** if it is a regular expression (in capital letters).

2 Task II (10% of total points of the exercise)

Two grammars are given. Your task is to write three valid strings and three invalid strings for each of them.

2.1 Syntax C

$$
\begin{aligned}
&\textit{start} \longrightarrow \textit{expression}\\
&\textit{expression} \longrightarrow \textit{term} \mid \textit{expression} - \textit{term}\\
&\textit{term} \longrightarrow \textit{factor} \mid \textit{term} * \textit{factor}\\
&\textit{factor} \longrightarrow x \mid y
\end{aligned}
$$

where * is a terminal symbol.

The three valid strings must be written into the file *exercise1/task2/**syntaxC/validC.txt***, one string per line. An example is shown in Figure 3.

Figure 3: Example of the file containing the three valid strings.

The three invalid strings must be written into the file *exercise1/task2/**syntaxC/invalidC.txt***, one string per line.

2.2 Syntax D

$$
\begin{aligned}
&start \longrightarrow sentence \\
&pathfilename \longrightarrow drive\ :\ \backslash\ pathname* \ filename \ fileextension \\
&drive \longrightarrow letter \\
&pathName \longrightarrow id\ \backslash \\
&fileName \longrightarrow id \\
&fileExtension \longrightarrow .\ id \\
&id \longrightarrow Letter* \mid Letter* \ Digit* \\
&Digit \longrightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \\
&Letter \longrightarrow a \mid b \mid c \mid d \mid e \mid f \mid g \mid h \mid i \mid j \mid k \mid l \mid m \mid n \mid o \mid p \mid q \mid r \mid s \mid t \mid u \mid v \mid w \mid x \mid y \mid z
\end{aligned}
$$

where * is the Kleene star symbol.

The three valid strings must be written into the file *exercise1/task2/**syntaxD/validD.txt***, one string per line.

The three invalid strings must be written into the file *exercise1/task2/**syntaxD/invalidD.txt***, one string per line.

3 Task III (30% of total points of the exercise)

A correct grammar and three correct examples of programs are proposed. The goal of this exercise is to create a parse tree and provide it in a JSON file.

3.1 Syntax E

$$
\begin{aligned}
&start \longrightarrow command \\
&command \longrightarrow program \\
&program \longrightarrow convert \mid run \mid comparison \mid exit\ now \\
&exit \longrightarrow shutdown\ -h \\
&convert \longrightarrow convert\ argument\ filename\ .\ extension \\
&argument \longrightarrow\ -RAWtoPNG \mid -PNGtoJPEG \mid -JPEGtoPNG \\
&extension \longrightarrow png \mid jpeg \mid raw \\
&run \longrightarrow python\ filename\ .\ py \\
&comparison \longrightarrow diff\ filename\ .txt\ filename\ .txt >\ filename\ .txt \\
&filename \longrightarrow letter*(letter|digit)* \\
&digit \longrightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \\
&letter \longrightarrow a \mid b \mid c \mid d \mid e \mid f \mid g \mid h \mid i \mid j \mid k \mid l \mid m \mid n \mid o \mid p \mid q \mid r \mid s \mid t \mid u \mid v \mid w \mid x \mid y \mid z
\end{aligned}
$$

where * is the Kleene star symbol.

A example of legal command is: `shutdown -h now`
A parse tree generated from this string is show in Figure 4.

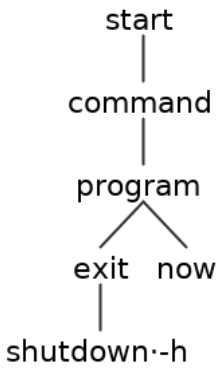

Figure 4: Example of the tree created from the `shutdown -h now` string.

The JSON file that can represent this parser tree is:

```
{
    "id": "start",
    "children": [
        {
            "id": "command",
            "children": [
                {
                    "id": "program",
                    "children": [
                        {
                            "id": "exit",
                            "children": [
                                {
                                    "id": "shutdown -h",
                                    "children": []
                                }
                            ]
                        },
                        {
                            "id": "now",
                            "children": []
                        }
                    ]
                }
            ]
        }
    ]
}
```

The goal of this exercise is to create the parse tree of the following three commands.
Please use the above JSON format to create the three requested parser trees.

1. `convert -RAWtoPNG newcar2019.raw`
2. `python helloworld.py`
3. `diff oldfile.txt newfile.txt > output.txt`

Write the parse trees into the following JSON files:

- parse tree of command 1 in the file *exercise1/task3/**command1/tree1.json***
- parse tree of command 2 in the file *exercise1/task3/**command2/tree2.json***
- parse tree of command 3 in the file *exercise1/task3/**command3/tree3.json***

4 Task IV (50% of total points of the exercise)

You are given regular expressions that describe the legal tokens of a language:

$$
\begin{aligned}
&\textit{keyword} \longrightarrow \textit{if} \mid \textit{else} \mid \textit{for} \mid \textit{while} \mid \textit{return} \\
&\textit{punctuator} \longrightarrow (\mid) \mid [\mid] \mid \{ \mid \} \mid . \mid ; \mid , \\
&\textit{comparison} \longrightarrow < \mid > \mid == \mid != \\
&\textit{assign} \longrightarrow <= \\
&\textit{op} \longrightarrow + \mid - \mid * \mid / \mid \&\& \mid \mid\mid \\
&\textit{id} \longrightarrow \textit{letter}*(\textit{letter} \mid \textit{digit})* \qquad (\textit{except for keywords}) \\
&\textit{number} \longrightarrow \textit{digit}* \\
&\textit{comment} \longrightarrow // (\textit{non\_newline})* \textit{newline}
\end{aligned}
$$

where $non\_newline$ means all characters other than $newline$ and $letter$ is a lowercase or uppercase letter. The $keywords$ are case-sensitive.

The aim of this exercise is to implement a scanner in Java that transforms a string in the language into a sequence of tokens, or reports an error if the string is illegal. Your must implement the scanner by hand, i.e., do not use a scanner generator tool.

The $comments$ must be skipped, so they don't appear on the output. All white space except for newlines must to be skipped. For example, the following two snippets of code correspond to the same sequence of tokens.

```
if(x > 0){
    y <= x;
}
else{
    y <= y / 2;
}

//This is a comment.
for(i <= 0; i < 10; i++){
    y <= 2 * y;
}
```

```
if(x>0){y<=x;}
else{y<=y/2;}
//This is a comment.
for(i<=0;i<10;i++){y<=2*y;}
```

An Eclipse project for the scanner implementation is provided: *exercise1/task4/*. You can import the project template (.zip) into Eclipse as follows: *File-Import-General-Existing project into Workspace-Select archive file-Finish* .
Please, implement your scanner in the method $public\ static\ List < String >\ functionScanner(String\ input)$, which you find in file exercise1/task4/scr/scanner/TokenScanner.java.

The input to the scanner is a $String$ containing a snippet of code. The output is a $List < String >$ with all the tokens found by the a scanner. In case of illegal tokens, the output must be a $List < String >$ with a single $String$ element: "Illegal".

You find some JUnit tests in folder *exercise1/task4/scr/scanner/*. Use them to check whether your scanner works. They can be run using Eclipse. We will use additional tests to evaluate your solution, and you are advised to also add additional tests for your own testing.

For the submission, your Eclipse project must be exported into a .zip archive using Eclipse: *File-Export-General-Archive File*, and then added to the tree structure in Figure 1.