UE COMPILER CONSTRUCTION **Summer Term 2025**

AAU Klagenfurt **Auer / Taschwer**

Assignment 2

Learning goals:

Apply the concepts and methods presented in chapter “Scanning” of the lecture to some exercise problems.

Presentation:

Please prepare the solutions for informal presentation in class, and fill in the online check list in ZEUS.

Task 1 Regular grammars

(a) Define a regular grammar for (arbitrary large) hexadecimal numbers starting with the prefix `0x`. You may use `hexdigit` as terminal symbol.

(b) Give an equivalent grammar consisting of a single, non-recursive production in EBNF.

Task 2 Regular expressions

(a) Define a regular expression describing block comments that start with “/*” and end with “*/” (`/*...*/`). You may assume that block comments are *not nested* (i.e. `/*.../*.../*.../*/` does not occur in input). You may use the terminal symbol `char` for any character that is neither ‘/’ nor ‘*’.

(b) Can *nested* block comments be described by a regular expression? Explain your answer.

Task 3 Transform regular expression to DFA

(a) Give a graphical representation of the deterministic finite automaton recognizing the language generated by the following regular expression, which describes certain roman numbers:

```
C ( (X? L)? X? X? X? | XC )
```

(b) Explain how the DFA processes input strings CXLX and CXXL. Are they accepted or rejected?

Task 4 Test scanner implementation

Test the scanner implementation of the MicroJava compiler provided in Moodle by writing a test program that writes the token codes recognized for a given text input file to standard output. Provide one or more input files covering all valid tokens of the MicroJava scanner as well as whitespace and line comments, which should be skipped by the scanner. Moreover, make sure to include invalid tokens in some input file, which should result in error tokens.

Task 5 Test scanner implementation

Extend your test program of Task 4 by supporting an optional second file name as a command-line parameter that contains the expected sequence of token codes (“ground truth”), allowing the test program to determine if the test case has passed. Provide a ground truth file for each test input file of Task 3 and make sure that all tests pass.

Task 6 Extend scanner implementation

Extend the scanner implementation of the MicroJava compiler provided in Moodle to implement the following added functionality:

> Allow block comments enclosed by `/*` and `*/`. You may assume that block comments are not nested.

Test your scanner implementation by extending your test cases of Tasks 4 and 5 accordingly.

Task 7 Extend scanner implementation

Extend the scanner implementation of the MicroJava compiler provided in Moodle to implement the following added functionality:

> Introduce a new token `**` that could be used to support the power operation in MicroJava ($x{**}y = x^y$).

Test your scanner implementation by extending your test cases of Tasks 4 and 5 accordingly.