Remarks:

- During the exercises we will implement our very own compiler for the simple language *WHILE*.
- The target code will be Jasmin : `jasmin.sourceforge.net`
- The following programs implements the gcd-algorithm in *WHILE*.

```
1       /* GCD-Computation of x and y
2           w/ WHILE */
3       int x; int y;
4       x = read();
5       y = read();
6       while ( x != y ) {
7           if (y <= x) {
8               y = y - x;
9           } else {
10              x = x - y;
11          }
12      }
13      // Output result
14      write("GCD: ");
15      write(x);
```

- Our *WHILE* programming language should capture variable declarations (ints only), assignments, arithmetic operations, conditional branches, loops, basic I/O and Java-style comments.

Exercise 1 (Lexemes, Symbol Classes and Tokens): (2 Points)

Give a complete list of the symbol classes and corresponding tokens needed for the lexical analysis of *WHILE*. Decompose the if-branch of the gcd-program (lines 7-9) into a sequence of lexemes and translate each lexeme into a symbol.

Exercise 2 (Simple Matching Problem): (3 Points)

a) Give regular expressions and languages for the keyword $while$, identifiers ids and comments $cmts$ in *WHILE*. Denote the languages by $\mathcal{L}_{\text{while}}, \mathcal{L}_{\text{ids}}, \mathcal{L}_{\text{cmts}}$.

b) Derive an NFA that accepts $w \in \mathcal{L}_{token}$, $token \in \{while, ids, cmts\}$ in state q_{token}.

c) Solve the simple matching problem for the input string $/*stup1d\,commen+*/$ using the algorithm given in the lecture. Apply the DFA- as well as the NFA-method.

Exercise 3 (DFA Implementation): (5 Points)

This course will be accompanied by a series of practical assignments with the goal to build our own compiler. Please consider the following general remarks regarding implementation assignments:

- All implementation tasks must be done in Java.
- Submitted code which does not execute is worthless (0 points). Therefore make sure you submit any third party libraries (jar files) that you have used and if you need any adaptation of the classpath in the javac/java call, then provide us the command line that will run your code. Of course, you should test you code, too.

- You may use third party libraries that do not simplify the assignment considerably. I.e. you might want to use some data structures from, say Guava[1], but you should not use some library that would solve the task automatically like lexer or parser generator libraries.
- Please document your code properly, such that it is possible to grasp the your ideas quickly.
- The code will be graded mostly by functionality but also by its clarity and style.

In this exercise we make the first steps towards building a lexer. The task of a lexer is to read an input string and return a sequence of symbols. For now, we start by building deterministic finite automata that recognise particular tokens. Please consider the hints below.

1. Write a class, say DFA.java, that is instantiated with a string. This class should represent a DFA that recognises exactly the given string.
2. Write a class, say CommentDFA.java, that recognises single-line and multi-line comments. (Note, that a single-line comment is terminated by a newline symbol \n (Linux), carriage return \r (Macintosh) or \r\n (Windows).)

For testing, instantiate CommentDFA and DFA(”while”). Let them both run on the following input words: ”while”, ”While”, ”/**while*/”, ”/* */*/”, ”//foo\n”. The expected output should be something like :

```
$java Main test1.txt
input: while
WHILE: accept
COMMENT: refute

$java Main test2.txt
input: While
WHILE: refute
COMMENT: refute

$java Main test3.txt
input: /**while*/
WHILE: refute
COMMENT: accept

$java Main test4.txt
input: /* */*/
WHILE: refute
COMMENT: refute

$java Main test5.txt
input: //foo

WHILE: refute
COMMENT: accept
```

Hints:

- It might be worthwhile to create an abstract class, say AbstractDFA.java, that implements the common methods of all your DFA implementations.
- On the course web page we provide you the Main.java file that contains everything necessary to read input as a string from a given file.

[1] http://code.google.com/p/guava-libraries/