apl. Prof. Dr. Thomas Noll
Sebastian Junges, Christoph Matheja, Matthias Volk

Compiler Construction 2017
— Programming Exercise 1 —

Upload in L2P until May 16th before the exercise class.

General Remarks

This course will be accompanied by a series of practical assignments with the goal to build our own compiler `i2Compiler` for the language *WHILE*. The *WHILE* language captures (integer) variable declarations, assignments, arithmetic operations, conditional branches, loops, basic I/O (read and write) and Java-style comments. Please consider the following remarks regarding implementation assignments:

- Programming exercises will be accompanied by a small framework including predefined classes and method declarations. Your task usually is to implement these methods. Please do not modify the signatures of the provided methods.
- Please document essential parts of your code properly such that it is possible to grasp your ideas. Although the code will be graded mostly by functionality, your comments will help us to clarify whether a bug is a conceptual mistake or just a small error.
- The practical part will be implemented in Java 8. You may use the standard library to solve the programming tasks. Other libraries are not allowed.
- Submitted code which does not execute results in 0 points. Therefore make sure you submit everything that you have used to run your code.
- Your solutions to the practical programming exercise should be uploaded via L2P as a zip file.
- If you have questions regarding the exercises and/or lecture, feel free to post in the L2P forum, write us an email at cb2017@i2.informatik.rwth-aachen.de or visit us at our office.

Programming Exercise 1 (5 Points)

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.

We provide you with a framework which contains the essential class definitions and method declarations for this task. The easiest way to work with that framework is to simply import the files as an existing project into Eclipse. Parts which need implementation are marked with `TODO`.

(a) Implement the class `AbstractDFA.java` in package `lexer` representing an arbitrary DFA with its states and transitions. You may use `Pair.java` in the `util` package.

(b) Implement the class `WordDFA.java` in package `lexer.dfa` which is instantiated with a string. This class should represent a DFA that recognises exactly the given string.

(c) Implement the class `CommentDFA.java` in package `lexer.dfa` that recognises single-line and multi-line comments.

For testing, you find several small examples in the `test` directory. You can compile the program with

```
$javac -d bin -sourcepath src src/Main.java
```

and then run the examples with

```
$java -cp bin Main tests/test1.txt
```

The results on the examples should be:

```
$java -cp bin Main tests/test1.txt
input: while
WHILE: true
COMMENT: false

$java -cp bin Main tests/test2.txt
input: While
WHILE: false
COMMENT: false

$java -cp bin Main tests/test3.txt
input: /**while*/
WHILE: false
COMMENT: true

$java -cp bin Main tests/test4.txt
input: /* */*/
WHILE: false
COMMENT: false

$java -cp bin Main tests/test5.txt
input: //foo
WHILE: false
COMMENT: true
```