Documentation of the 1. Project Implementation for IPP 2020/2021
Name and surname: David Mihola
Login: xmihol00

# Implementation Details

This document describes the logical structure of the parser script source code and the approach used to accomplish the assigned task. The more technical details (f.e. implemented functions) are described directly in the source code.

## Command line arguments

I did not use any library for parsing the command line arguments, as there is required support for only one `--help` argument in the standard assignment. Additional arguments used in the assignment's extension are also in a format not satisfiable by any library, which I am aware of. These arguments are parsed in a class `statistics`, about which below.

## Parsing

The parsing is performed one instruction at a time, therefore the source code is read by lines. The read line is then trimmed from any white characters and is separated to an array of lexemes. Parsing of a lexemes is proceeded by the use of regular expressions and a very simple recursive descent parsing algorithm. All of the regular expressions are predefined at the beginning of the document. Text strings are parsed separately by a function `parse_string()`. The first lexem on a line must represent an instruction opcode. The opcode determines, how many other lexemes and of which format are to follow. That results in a appropriate parsing functions being called on each lexem. All lexems are firstly checked for a start of a comment or for a trailing comment, if such a lexem appears, the rest of the lexems on the line is skipped. Comments can also appear on separate lines, these lines do not start with an instruction opcode and are skipped entirely.

## XML Construction

The XML output is created by the use of the DOM library. The usage of the DOM library is encapsulated into a class `XML_builder`. An object of this class is responsible for initiating the XML structure, keeping track of the instruction order and the order of each instruction's arguments. The parsing algorithm calls appropriate class methods to add an instruction – `create_instruction()`, an instruction's opcode – `add_instruction_opcode()`, an argument – `add_instruction_argument()` etc. When the parsing is completed successfully, the constructed XML object is written out to `STDOUT`.

## Statistics Extension

The collection of statistics is encapsulated to a class `Statistics`. An object of this class offers methods for parsing the command lines arguments determining the desired statistics to be collected and written down to a given files – `parse_arguments()`, counting the number of parsed instructions – `add_inst()`, comments – `add_comment()`, jumps – `add_jump()` etc. When a class method for counting the number of jumps is called, it is also evaluated, whether the jump is forwards or backwards, based on the already collected label names. If the label, which it is jumped on, is already known, then it is a backward jump, otherwise it is an undecided forward jump. When a newcoming label matches an undecided forward jump(s), it is declared as decided. At the end of execution the number of undecided forward jumps is used as the number of bad jumps. The collection of all statistics is performed at every execution of the script, but it is used only when specified by the command line arguments.

## Testing

The parser script was thoroughly tested by the samples included in the `tests` directory and additionally by the tests offered on the faculty Discord channel.