Implementation documentation for the 1. task for IPP 2018/2019
Name and surname: Sabína Gregušová
Login: xgregu02

Introduction

My task was to create a script in PHP for lexical and syntactical analysis of language IPPcode19 derived from IFJcode18.

My Solution

I started implementing `parser.php` as soon as possible, so that I would have more time for other projects during the semester.

Firstly, my script `parser.php` checks the input arguments. There is only one possible argument in the basic script '–help', which cannot be combined with anything else. It continues with opening file that was given in `STDIN` or accepting input from `STDIN`. After successfully opening/accepting the input, the script reads the first line to confirm that it has the correct header. If the header is correct, I initialize memory for the `XML` output by using functions from `XMLWriter`, which formats the output accordingly. The script only outputs the `XML` document if no error was encountered during lexical and syntactical analysis and all error information is written to the `STDERR`.

The input is read line by line and I've used regular expressions to extract the opcode and check whether it's acceptable. Since the opcode is case insensitive, I've used function `strcasecmp` in to confirm that the opcode is correct. Since opcodes can accept zero, one, two or three operands, I grouped them together and created function for each of these groups. Therefore, I implemented functions that handle zero, one, two or three arguments and each of them checks whether the argument type matches the required type(s) for given opcode. I also check for too few or too many operands for given opcode. My regular expressions can extract the required information with multiple white spaces, but the rule is, that the operands and opcodes need at least one whitespace between them.

Each argument is checked lexically by function `evaluate_arg`. I've decided to do as many checks as possible, so I make sure that following input is correct: type of frame, type in front of @, correct value after @ for given type, acceptable variable name, acceptable string including the escape sequences. I've implemented function `write_arg` in order to make writing the arguments in `XML` easier and quicker. It takes order number of argument within one instruction, formats it and sticks it to the end of my global `XML` string.