Documentation of parse.php for IPP 2019/2020

Name and surname: Zhamilya Abikenova

Login: xabike00

1 Parse.php

1.1 Parser implementation

The script *parse.php* was implemented in PHP 7.4. It reads the input code received from standard input. This input code has to be written in the *.IPPCode20 language*. As such, the header needs to contain this information on the first line of the code, or immediately following empty lines or comments. If the header is present, the script starts scanning the standard input line by line and parsing each one at a time.

Syntactic and lexical rules are being checked inside the *class LineClass*. First, we tokenize the scanned line into instruction and argument tokens in the method *parseLine*, and further split the argument token into the data type and the value of a variable or constant. Those split values are then stored into the arrays *$par_type* and *$par_value* respectively.

When the line is tokenized, the instruction token is compared in the method *_operandCounter*. If it exists, it is then further tested for the correct number of arguments (i.e. *RETURN* having 0 arguments and *READ* having 2). In the case of *READ*, the method *_operandTwo* is called, which both checks for the correct number of arguments but also the correct data types and values of every one of them. (i.e. *READ* <*var*> <*type*> where <*var*> is a legitimate variable such as *GF@counter* and <*type*> is a string from the static set of *{string, int, bool, nil}*. In the case of strings, the parser also ensures their value is properly edited to match XML rules in the case of sensitive characters {<&>}. This is all ensured via regular expressions and their functions *preg_match*.

If all conditions pass, the object then creates a string in the format of XML via the *genXML* method. This string is then further passed to a string named *$xml_string*, which appends every single line worth of generated XML code. If all lines of input code were parsed without error, where the string is printed out to standard output. Otherwise, the function *errorCode* is called which exits the code and prints the error's number on standard error output.

The script also supports the use of *--help* argument, which gives a brief description of it's usage.

1.2 Bonus implementation

Extension ***STATP*** collects statistics based on given arguments and writes statistics to a given file. For each argument there is a variable that increases by 1 ($jumps, $comments and $labels). At the end of the program, we list all required statistics in a file in the requested order:

--*loc* - variable *$loc* is assigned to a variable *$counter* that stores information (number) about all instructions that contains an input file;

--*jumps* - for each jump instruction *$jumps* is increased by 1;

--*comments* - for each comment *$comments* is increased by 1;

--*labels* - all labels stored in an array and the number of unique labels is counted.