Task documentation SYN: Syntax highlighting in PHP 5 for IPP 2013/2014
Name and surname: Marek Milkovič
Login: xmilko01

1 Assignment

The task was to create a script in PHP 5 to highlight syntax in the input text based on the regular expressions written in the format file. Syntax highlighting was achieved with HTML tags which were inserted into the input and printed out on the output.

2 Design

2.1 Error codes

To effectivly use all advantages of PHP, I have decided to throw exceptions from the code rather than returning error codes accross the whole application. Every thrown exception in script carry the integer representing the exit code for the script and also message which was used for debugging purposes during the development process.

2.2 Regular expressions

Since regular expressions have used unique syntax in this assignment, it was needed to take measures how to evaluate these expressions. There were two possible solutions from my point of view.

1. Build a system that can read this expression and dynamically construct automaton that can accept input strings conforming regular expression
2. Build a system that can read these unique syntax of regular expressions and translate it into PHP regular expressions

I found the second solution less time consuming than first and made finite automaton that reads the regular expression from the format file and translates it.

2.3 Argument parsing

PHP has builtin support for parsing arguments with function `getopt` which would save my time, but the behavior which I needed didn't require the short and the long format of arguments like the function `getopt` offer, so I decided to write my own parser, which is capable of throwing the exceptions of my need.

2.4 Inserting tags

The problem of inserting tags in the input text with preserving order specified in the format file while respecting the fact that closing tags shall come before opening tags can be solved by many approaches.Direct editation of the input text is not clever, because then the problem with ignoring the inserted tags while matching regular expressions come into the way. You also

have to think about the order of closing and opening tags.

The concept looks like this:

- Store the positions in the input text where some tags are being applied
- For evey position, store the list of closing tags and the list of opening tags
- Upon printing out the output, traverse first through the list of closing tags in reverse order, then through the list of opening tags in regular order

3 Implementation

The code is divided into 4 files.

- `syn.php` - Main file for running the script
- `ArgParser.php` - Contains class for parsing the arguments from the command line
- `Formatter.php` - Contains class that does the formatting of the input text, also reads the input and prints the output
- `Tag.php` - Class for string the tags in more abstract way

The script starts in `syn.php` where the main part of the code is enclosed in `try - catch` block to catch exceptions. `ArgParser` class is then used to recognize whether the input arguments are valid and if they are valid, they are also parsed. Upon successful parsing, the new `Formatter` is created with the paths to the input, the ouput and the format files.

Formatter reads the input file (or `STDIN` in case of not `--input` argument) line by line into the memory. When it reaches the end of input, the reading of the format file starts by calling the `Formatter` method `ReadFormatFile`. It reads the format file line by line and calls the method `ParseFormatLine` which uses regular expressions to divide the regular expressions specified in the format file and the format types for this expression. The format regular expression is translated to the PHP regular expression with method `TranslateRegex` which implements finite automaton. The format type part is parsed again with method `ParseFormatType`.

On successful parsing of the format line, the translated regular expression is used to match the input text and the results of the match are stored in the associative array with the key as the offset of the match and the value as the array carrying two another arrays, one for closing and one for opening tags.

When all format lines were parsed and no error occured, the associative array for storing tags is sorted by the key in asceding order. This array is then traversed the way that the first non-visited key in the array is taken and eveything between the previous offset (which is set 0 at the beginning) and the current offset (value of the key) is printed out to the output file (or `STDOUT` if no `--ouput` argument entered), then the array of closing tags is traversed in reverse order and every tag is written and after that, the array of opening tags is traversed in regular order and also written to the output. Then the next key in the associative array is visited.