Task documentation SYN: syntax highlighting in PHP5 for IPP 2013/2014
Name and surname: Ivan Ševčík
Login: xsevci50

Task specification

The task was to create a script in PHP5 language that would highlight syntax of inputted text (either through command line or file) using rules defined in format file that describe how to transform input into formatted text. Each rule is defined as regular expression that matches certain text which in turn is placed into one or multiple pairs of formatting tags defined by styles following regular expression. Each line of format file represents such rule. The order in which are formatting styles applied depends on their appearance in format file – from left to right and from top to bottom. Closing tags have the exact opposite order, therefore the last opening tag is closed as first. The regular expressions are defined as an extended version of what was used in the IFJ course. They are not listed here because of space constraints and for the same reason the description of styles was omitted. Available switches with description can be printed with `--help` switch when running the script.

Proposed solution

The idea is to convert all regular expressions from IFJ course format into something that can be used as regular expression for PHP functions without changing what's matched by such expression. For most of symbols there exists a direct representation, such as + and `*`. Also most of characters that represent a basic regular expression are used the same way in PHP. However, some PHP special symbols without any meaning in IFJ will have to be escaped using `\` and similarly some IFJ symbols have to be converted. One such pair of symbol is `{` and `.` where curly brace should have no special meaning and will be converted into `\{` and `.` has a different meaning – instead of concatenation it matches any character in PHP, therefore to mean concatenation it will be left out in process of conversion.

After converting all regular expressions they can be applied in sequence as they appear in format file. However input shouldn't be changed as that could prevent next regular expression to match. The solution is therefore to somehow just mark where the match was found and after applying all regular expressions print out the input interleaved by formatting tags using marked positions. For such method a stable sort will be needed to make an ascending order for positions of tags, but not changing the order of tags at the same position. This also causes problem that closing tags will appear in the same order as opening tags and not the opposite. This can, however, be fixed by first placing the closing tags on a small stack, that will be emptied after moving to next position or in case the next tag at the same position is opening one.

Implementation details

First of all, there was a need to support all command line arguments to be able to properly test the behavior later. For this purpose a finite state machine (FSM) was created that can parse through arguments and detect error. Another FSM was used for conversion of regular expressions. Processing one character at time, it needed several state variables that would modify how the next character will be interpreted or signal errors when incorrect regular expression was used as an input. This also laid a base for internal design and most of classes that were needed later on.

To match and find all occurrences of regular expression in the input, `preg_match_all` was used with modifiers (flags) `PREG_OFFSET_CAPTURE` and `PREG_SET_ORDER` that caused each match to appear as one item in returned list that contained both the matched string and its starting position. To mark the position for later use, a class `Tag` was used that can hold information about position, formatting style and if it's opening or closing tag. Iterating through all regular expressions, an array was filled with tags that were then sorted.

By sorting the tags, the formatting part was completed and it only needed to be printed to the output. Interleaving was done by first printing number of bytes preceding next tag and then printing all tags at the same position. This was placed into loop, printing out all tags mixed with input, and only the rest of input after last tag had to be printed out after loop.

During implementation, several problems arise to proposed draft. First of them was not accounting for multi-byte UTF-8 characters required in original specification. This has later shown up not to be such a big problem as there exists a switch to enable UTF-8 in PHP regular expressions. Matched positions are still incorrect in respect to number of characters, but this didn't cause any further problems as the correct number of bytes is printed together and the representation of file is solely the role of an editor in which the file is opened.

Next complication was caused by PHP5 not having a built in stable version of sort. To not break the draft, it was easier to implement a merge sort that is both stable and relatively effective, but using a sort written in an interpreted language wouldn't be a good decision for professional application. As a workaround, it could've been written in compiled language into a library and called from PHP.

Conclusion

A script providing required functionality was created along with this documentation. Being evaluated with difficulty 1, this task has proven to be relatively difficult. This was caused by lacking built-in tools in PHP5 that would provide elegant solution, such as stable implementation of sort, insert operation on top of string class and native support of multi-byte characters as special functions and switches had to be used. However, the task itself was creative and interesting, providing enough space for many different implementations.