Task documentation CST: **C Stats in PHP 5** for IPP 2014/2015

Name and surname: **Daniel Dusek**
Login: **xdusek21**

Whole script is written with maximal usage of object-oriented-programming version of PHP. OOP version of PHP is used because of its advantages such are encapsulation or possibility to re-use the code. And of course, from symbolic reasons - course is named Principles of Programming Languages (and OOP).

Object Oriented Design of CST filter

The CST filter is realized with a help of two classes - `IOHandler()` and `cstFilter()`. Neither of two classes inherits from another, but class `cstFilter()` uses public methods provided by `IOHandler()` class on several ocassions.

Class IOHandler()

As the name almost suggests, this class is resposinble for all input and output communication and handling. Methods contained in the class covers all work with input and output starting with showing the help, going through loading input file, ending with writting to stdErr and terminating filter with return code.

Class cstFilter()

Public methods of the class are used by cst.php script that is executed when running the filter. Main method here is named Run() and basically Runs the filter, using classes's private methods. Checking if given parameters can be used is covered in this class, so is every other method that filters input.

Parameter processing

All input parameters are processed with php library function `getopt()` designed specifically for this task. Despite the fact the function has its weak spots, for the purposes of this project it was the only logical choice.

Handling input data

As was mentioned earlier, filter cst.php is designed in an object oriented way. One of total two classes is named `IOHandler()`. Methods of this class are responsible for most of the input data checks - methods check if given parameters are all right,

Filtering input data

Methods included in `cstFilter()` class are responsible for practically everything that has something to do with filtering input data.

One of the major issues in the project is to correctly remove macros, commentaries, string and character literals. Though this could seem like an easy task for regular expressions, it has proven to be much more complex to solve. In general, regular expressions are not so powerfull tool when there is a need for knowledge of context, which in this case is. Therefore the author decided to use final state machine to solve this issue.

Final State Machine

Removal of commentaries, macros, character literals and string literals is realised as a final state machine that determines six significant states as follows: `sSTART, sMACRO, sBLOCKCOMMENT, sLINECOMMENT, sSTRING, sCHARLIT`. Those states are intentionally named to be self-explanatory.

At the beggining the file content is stored in a string variable and then the string variable is parsed char by char. FSM decides to change or not to change the state in dependence on the character that is read from the string variable. Based on state we enter, different actions are provided - for instance, if the commentary is entered (or is stayed in) the commentary character counter variable value is increased by one. Then the character belonging to the commentary is removed, because for other functions it is required for commentaries to be purged from input.

Parameter -w and no FSM

While every other switch needs commentaries, strings, chars and macros to be removed, -w switch that looks for the specified string even inside those mentioned, there is no need to remove them. That really simplifies the whole task to a need to apply one simple regular expression to count all the appearances. For demonstration of simplicity of the task, the regular expression is lited here:
`/$options[w]/`
Where `$options[w]` stands for string filter is searching for and `//` stands for delimiters used by `preg_*` function.