Task documentation CST: C Stats in PHP5 IPP 2014/2015
Name and surname: Dávid Mikuš
Login: xmikus15

1 Task

The task was to create script in PHP 5 to analyze source file in C language. Subsequently print out statistics of comments, identifiers and keywords.

2 Design

2.1 Error codes

Instead of every time checking return value of method, an exception is thrown with integer value representing error code and message which contains info about error. If an exception occured, then the script is terminated and error code is returned.

2.2 Argument parsing

Instead of using PHP builtin method `getopt`, we decided to write own argument parser. Main advantage is that we have everything under our control, so we can throw an exception when it is needed. For example when unknown parameter is found.

2.3 Analyzing source code

Structure of source code is complex, it is not so easy to match everything correctly according to C standard. From our point of view there were 2 solutions how to solve this task.

1. Write a custom parser and do semantic analysis.
2. Use regular expressions and finite state machine.

First solution is better one, we could match everything correctly but it is time-consuming. We chosen the second one, because it was easier to implement. But this solution is not so clean as the first one. It can not match everything correct and we have to take advantage of regular expressions.

3 Implementation

The code is divided into 3 files.

- `cst.php` - Main file for running the script
- `ArgParser.php` - Includes class for parsing arguments
- `CAnalyze.php` - Includes main class for analyzing source code

3.1 Main script - cst.php

The script starts in `cst.php`, every statement is enclosed in `try...catch` block. If an exception is thrown, it is caught, script is terminated and returns error code, else arguments are passed to object from class `ArgParser` which is constructed. Afterwards are arguments parsed and checked if they are valid, otherwise an exception is thrown.
If argument `--input` was specified and it is directory, object `RecursiveDirectoryIterator` is constructed which recursively traverse through subfolders. If script was ran with argument `--nosubdir` then it process just files in folder. For each `.c` and `.h` file is created new object from class `CAnalyzer`. Every file and result is stored in associative array. If argument `-p` was specified then path to file is stripped and remains just file name.
At the end, every item is print out to `stdin` or if `--output` was specified then script open the file and print statitics to this file.

3.2 Source code analyzer - CAnalyze.php

In constructor file is opened and its content is saved to variable. If file is not readable the exception is thrown. At begin it replaces `\r\n` with `\n` to have uniform new line.
Before analyzing macros, comments and string literals have to be deleted. Macros are deleted by regular expression. Comments and string literals by finite state machine(FSM), because we have to look on context. In this FSM is created new variable which contains file content without macros,comments and string literals. In addition to it, it also counts comments.
Based on arguments next operation is chosen.

- -c - Return how many chars was found in comments, it was counted at begin in FSM
- -w=pattern - Return how many occurences of pattern was found in file. It's counted by regular expression. To avoid misconception by using regular expression, we have to escape special characters. This is done by PHP builtin method `preg_quote`.
- -i -k - Both options are done by the same FSM. FSM find sequences of characters which can represent identifiers. Afterwards, the function `isKeyword($id)` is called. If it returns true then keywords counter is incremented, otherwise identifiers counter is incremented. At the end -i returns identifiers counter and -k returns keywords counter.
- -o - Returns how many operators was found in file. For this purpose is used regular expression. But not in every context character which can represent operator is operator. Characters `+,-,.` can be in number literals. In this case it is not operator. Next one, character `*`, it could occur in declarations, which doesn't represent operator. To solve this problem is used another regular expression which match number literals and declarations. Afterwards in these matches is counted how many times occured these characters and substracted from final count.