Task documentation CHA: C Header Analysis in PHP 5 for IPP 2016/2017
Name and surname: Dávid Bolvanský
Login: xbolva00

1 Basic Idea

As an initial idea, we decided to use regular expressions as the base for C Header Analysis. We put modifier **u** to any regular expression to active and support UTF-8. Our final implementation is based mainly on regular expressions in our parsing logic. The project can be divided to the following phrases which are described below:
→ Processing Arguments
→ Input Source
→ Header Analysis
→ Output Source
→ Writing to XML

2 Processing Arguments

We used the combination of processing **argv** and **getopt** to detect options and their values, if any. At first we supported standard options, later we added supported for shortened options like **-i, -o**. In this phrase we check entered options for various errors. Some options require integer values, so we check if entered values are valid integer numbers. Also, input and output cannot be empty strings. We also check arguments for duplicated options. If any error, an error message is printed and script exits with code 1.

3 Input Source

As an input we can have a file or a directory. If neither of them is not specified, current working directory is used. If the input is type of directory, we recursively find and analyses files with extension **.h**. If input is specified via –**input=file** but is not valid (input file is not readable, does not exists, is a directory, etc. . .), an error message is printed to standard error stream and the script exits with the error error 2.

4 Header Analysis

As we mentioned, our analysis of function definitions and declarations is based on regular expressions. The filtering, parsing and analysing logic is in **extract_headers_for_file**. At first, we load the content of the file to a string. This string is later filtered. We remove the comments, macros, enum, struct definitions, typedefs, string literals. Later, we process this string line by line - we remove lines which cannot contains function definitions or declarations, this is also done by using regular expressions. We find functions using our specific regular expression via **preg_match_all**. When we find a function, we put its name to list of already analysed functions. If there is only a keyword **void** in the function parameters list, we recognize this function as a function without any parameters. If we find three dots (**...**) at the end of function parameters list of the certain function, it means that the function has variable length arguments. If –**no-duplicates** is set and we find name of current found function in that list, we skip analysis of this function. We also skip analysis if a return type of function contains inline keyword and –**no-inline** is set. If –**max-par=n** is set, we skips function which have more than **n** parameters. We remove whitespaces from return types and parameters using function **normalize_spaces** if –**remove-whitespace** if set. Function parameters are parsed in function **parse_params_list**.

5 Output Source

The output of C Header Analysis can be saved to file. If –**output=file** is not specified, output is printed to standard output stream (stdout). If output is specified but is not valid (output file is not writable, output file is a directory, etc. . .), an error message is printed to standard error stream and the script exits with the error code error 3.

6 Writing to XML

At first, we were manually printing XML elements using **fwrite** but later we decided to use a **PHP** library **XMLWritter** which provides useful APIs above XML I/O operations. Also it provides functions which we used to implement some functionality of this script, namely argument **pretty-xml** and we used related XMLWritter's functions **setIndent** a **setIndentString**. We define output for **XMLWritter** using **openURI**.

7 Extensions

We implemented both extensions in this task.

7.1 FUN (1.0pt)

To implement this extension, we cannot rely on our basic regular expresion to find functions definitions/declarations so we had to introduce additional regular expressions to analyse more complex functions, which for example return a pointer to a function. Also, we had to change previous logic to parse function parameters since we cannot use comma as delimiter anymore, since as part of this extension, we can have function pointers as parameters - e.g. **void foo(void (*f)(int a, int b))**. As a solution, we used simple brackets counting in parameters so we were able to determine whether the incoming comma seperates parameters in functions or this comma is part of a function pointer presented in function parameters.

7.2 PAR (1.0pt)

This extension was slightly easier to implement than FUN, since we had to change only a few things in regular expressions, but later we found potentional issues. When we analyse e.g. function **void foo(const float)**, we had to change our parsing logic of parameters which checks now if a string, which can be name of argument, does not contains name of some data types in C so we adjusted parsing code according to this information.