Task documentation: CHA: C Header Analysis in Perl for IPP 2011/2012
Name and surname: Michal Lúkáč
Login: xlukac05

Introduction

This program does the analysis of functions located in c-header files and print statistics about functions in xml-format to specified file or standard output. The implementation is in Perl programming language which is strong for text processing, bioinformatics analysis and basic scripting. Perl has also good regular expression engine and modules/libraries.

Arguments processing

If you do not know how to use this script, run program with --help switch. For script arguments, program uses Getopt::Long module in procParams subroutine. You can specify the input file or directory with --input='fileordir' switch. If this parameter is not entered program searches and analyzes all header files in current directory and his subdirectories. Program uses File::Find and Cwd modules for searching files. It all happens in subroutines *searchDirs* and *getFiles*, where filenames are push to array *@G_IFILES*. Another argument is --output='file' which determines output file. If not specified the output is printed to stdout. There are also arguments for removing white spaces --remove-whitespace implemented by regex or --no-inline for blocking inline functions. The flags($G_INLINE,...) are set if the arguments are entered.

Implementation

After what are arguments processed and flags were set, the main subroutine programm goes through all files from array @G_IFILES with foreach cycle. In this cycle I open and read input file to array @text1 and then join chars with join('',@text1); to $text. Xml-root element is printed. Then this text from file is send to prepareText which returns 'clean' text. As first, script removes all unnecessary things like commentaries, macros, strings, brackets, structures and variables. There are useful regular expressions(regex/regexs) for match and remove commentaries s/((?:\/*(?:[^*]|(?:*+[^*\/]))**+\/)|(?:\/\/.*))//g and s/\/\/[^\n]*//g. For removing strings I use s/"[^"\\]*(\\.[^"\\]*)*"//g. For removing blocks of code in curly brackets I use simple algorithm in which I count number of brackets. Clean text with functions is send to subroutine findFunctions. With simple regex [\w\s]+[^{;]*; subroutine returns array of functions. Inline functions are removed from this list if global flag/variable G_INLINE is true and \sinline\s or ^inline\s are matched. In programm I use foreach cycle to iterate all functions. Each function is passed to subroutine getParams which parse parameters of function with simple algorithm and five regexs. Parameters are returned as array to programm where are further processed. With handled parameters, 'params' part of function like (int a); is removed and then passed to in-build reverse function for string. After that it is simple to use regex \w+ to get function name. Similar approach is used to get return type of function. If --no-duplicates is set, program iterate all functions name which are saved to array and try to compare with actual function name. If function is not in array I use push(@functionsname,$funname); to add it. If function name is already in array, variable $isdupli is set to true. Otherwise is set to false. Next part is printing parameters information of function. If --remove-whitespace is defined, parameter is send to removeSpaces subroutine, which removes unwanted white spaces. I use regex commands for substitution like $text =~ s/\s/ /g; to replace any whitespace for blank space. Final part is to print xml-tag with parameter type and order number. There is also additional subroutine for printing whitespaces printWSpaces if --pretty-xml=k is set. Constant k set the number of blank spaces before xml-tag and one newline character after xml-header. Subroutine printWSpaces has parameter for level of depth.

Most of the parsing, replacing, searching is implement by regular expression. Program often use match(m), replace(tr), and substitution(s) regex commands with g-switch, which means 'apply globally'. Script exploits any xml module. All output is print directly to stdandard output or file. Script is divided into

subroutines for better overview. Subroutine exitError is call where there is any error in program like not correct parameters, error with opening/closing file or unexpected error. My script can also handle input function with parameters like “char *argv[]”, where output prameter type is “char *[]”.