Skills Measurement and Source Code Analyzer in C Programming

Nattapatch Srirajun[1]
and Somjin Juntarajessadakorn[2]
Nakhon Pathom Rajabhat University, Thailand
[1]nattapatch@webmail.npru.ac.th
[2]somjin@webmail.npru.ac.th

***Abstract* - Generally, source code analyzer is a tool for examining source code for a set of common defects and violations of good programming practice. This research aims to enhance this tool and evaluate the level of programming skill based on experts' skill. Our proposed tool consists of 2 modules: an analysis module and a measurement module. In the analysis module, there are 3 methods for analyzing programming skills which are memory usage, lines of code, and Big-O notation together with a technique of string scanning. Meanwhile, 95% confident interval based on the experts' skill is used as a baseline and measurement of the programming skills and performance of the students into 2 levels: good and excellent. Thus our tool can help students to self-assess for improving their coding.**

***Keywords* - Source Code Analyzer, Programming Skill, String Scanning, Confident Interval**

I. INTRODUCTION

According to programming, source code is analyzed by a compiler and an interpreter based on syntax of programing language. Including Context-Free Grammars (CFGs) and Regular Expressions (REs) are patterns that are used for checking the syntax of programing language. When developers want to solve a programming problem, they can find only correctness of the output, but not an algorithm. String scanning on source code is a technique that is adjusted for increasing usefulness. There are 3 techniques of string scanning on source code that consist of reengineering code, making queries on programs, and understanding programs [1]. Therefore, the beginner practice concerns only the suitable source code for solving the general programming problem.

This research proposes an alternative tool for any developer to improve their programming skill, particularly, computer science students. Our proposed tool consists of 2 modules: an analysis module and a measurement module. In the analysis module, there are 3 methods for analysis of programming skills which are memory usage, lines of code, and Big-O notation together with a technique of string scanning. Meanwhile, 95% confident interval based on the experts' skill is used to be a baseline and measurement the programming skills and performance of the students.

II. LITERATURE REVIEWS

Normally, software application is developed by using an Integrated Development Environment (IDE) tool. It can help checking syntax of programming language with a compiler or an interpreter. The principle CFGs and REs are basic for checking syntax programming. On limitation of the original theory of REs is that it concerns only the recognition of a string pattern for syntax of the programming language, without regard to other internal structure. Many researchers proposed some techniques for improving the potential of the compiler on REs by creating a compiler tool for improving compiler performance [2-3]. The research in [4] proposed Parsing Expression Grammars (PEGs) describes machine-oriented syntax. The syntax can reduce syntax ambiguity and increases simplified syntax definitions by making unnecessary and hierarchical components of lexical. PEGS can build with a linear-time parser avoiding both the complexity and fickleness of LR parsers. In addition, the pattern-matching tool based on PEGs for the Lua scripting language (LPEG) [5] was proposed with a small and efficient implementation of PEGs for pattern matching. The research of S. Paul, and A. Prakash [1], proposed a framework in which pattern languages are used to specify interesting code features for extending the source programming language with pattern-matching symbols. SNOBOL patterns and integration of regular expressions [6] were proposed to string scanning control for improving power and flexibility sufficient to handle complex and dynamic pattern contexts when they cannot be matched by regular and context-free languages.

In addition, source code analyzer is another technique that is used for increasing performance by adding to the REs mechanism. In the research [7], timing performance data is analyzed with searching single keyword pattern matching problem technique on various metrics to estimate algorithm performance. Moreover, Estimation of distribution algorithms (EDA) [8] was proposed with stochastic search methods that look for optimal solutions by learning and sampling from probabilistic tools on applicable to runtime analysis. The result showed easily on upper bounds on the expected runtime. K-scope [9] is a source code analysis tool that can be used to improve code performance for Fortran 90 and FORTRAN 77. The source code is simulated with Java to visualize on program structure by using a graphical user interface.

III. PROPOSED TOOL

Our proposed tool consists of 2 modules, which are the module of analysis methods, and the module of measurements and evaluation. These modules are created by the experts' submission their source codes to the system. Then the process of analyzer and measurement run and build up the training tool. This tool is used for testing and evaluating the programming skills and the performance in logical thinking, especially for students. Thus our proposed tool has 2 parts to work with the core module; the part of experts and the part of students. The part of experts is building up for the baseline, and then uses it for analysis and measurement in on the part of the students. For example, in the part of the experts, the system sums up all the memory usage of each expert and then passes the mean and value to the function of the confidence internal to create the critical margin for acceptance or rejection of the validity of the mean. In the part of the students, the overall memory usage in his/her code is computed in the same way of the experts part, and then evaluated whether the mean value falls into the experts range based on the confidence interval or not. Fig. 1, shows the system overview in which details of the core modules are following.

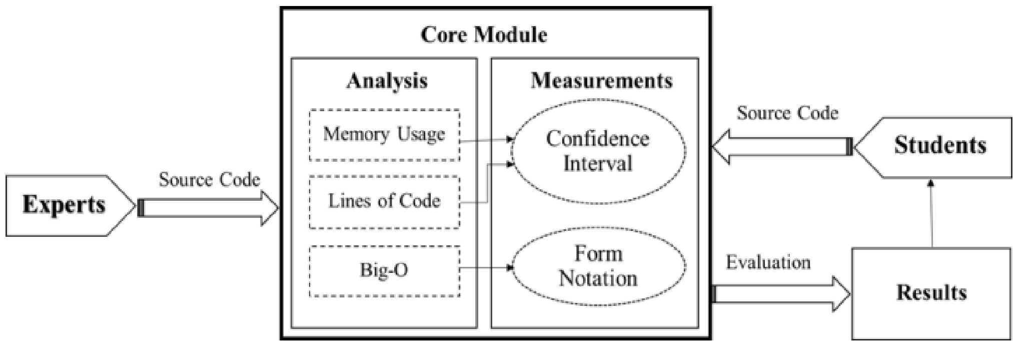


Fig. 1 System Overview

A. Analysis Methods

This module is used for analyzing the way of coding and creating the algorithms that they use for solving the problems. There are 3 main functions together with the technique of string scanning as follows:

1) ***Memory Usage Analysis***

This function is looking for the number and the type of variables that are declared in the code. The memory usage is calculated on runtime in which each variable is reserved bytes in memory. It is noted that each variable type is different size in memory. For instance, the variable type of "char" is 1 byte, "int" is 2 bytes, "float" is 4 bytes, and "double" is 8 bytes etc.

2) ***Line of Code Analysis***

This function is counting the number of lines of code in the program. Also in C programming, the number of line is counted on all the functions e.g. main(), and counting every command line ending with semicolon (;) within the functions.

3) ***Big-O Analysis***

This function is used to classify algorithms according to how the running time grows as the input size grows, in the form of growth rate functions. Thus, the efficiency of the algorithm regarding to the relation of time and size is compared to find the least time. Here are some example functions which are sorted least time in descending order: 1 > log n > log2 n > √n > n > n log n > n2 > n3 … nk > 2n , cn > n!.

B. Measurements & Evaluation

This module is the next step after analyzing the modules in earlier steps. There are 2 functions in the measurement and evaluation as follows:

1) ***Confident Interval Measurement***

This function is used for identifying and evaluating the performance of students' programming skills based on the experts' confident interval. Our methods assume that the values of the memory usage and the line of code are normally distributed in a reference population of experts. According to our proposed tool, the first step of the part of experts has 2 processes. The process of measuring and evaluating the memory usage, and the process of measuring and evaluating the line of code. Suppose the system runs the first measurement and evaluation process; the process of the memory usage of 36 experts (n=36), and calculate the sample mean (M) is 62 and the sample standard deviation (SD) is 30. Then M is a point estimates of the population mean of memory usage. The system is set to seek a 95% Confident Interval (CI), which is an interval estimate that indicates the precision or likely accuracy, of a point estimate. The 95% is the confidence level, or C, of the CI. The CI will be a range centered on M, and extending a distance (w: width) either side of M, where w is called the margin of error. The margin of error is based on the SE, which is a function of SD and n. In fact, SE = SD/√n = 30/√36 = 5, and w is the SE multiplied by $t_{(n-1),C}$, which is a critical value of the t statistic that depends on the chosen value of C. For C = 95, the value of t, with df = n –1 = 35, that cuts off the lower 2.5% and upper 2.5% of the t distribution; this critical value is 2.03. The margin of error is w = 2.03 * 5 = 10.15. The lower limit of the CI is M – w = 51.85, and the upper limit is M + w = 72.15, and so the 95% CI is (51.85 to 72.15), also written as (51.85, 72.15). This is the interval estimate of the mean of memory usage based on the experts. Moreover, the process of measuring and evaluating the line of code does in the same manner of this process.

In addition, we categorize the performance level into 2 levels: "excellent" and "good". The excellent performance level is calculated by the mean value falling into the range of M $\pm$(w/2); in this case the range is 62$\pm$ 5.075, and the good performance level is over the range of M ±(w/2) but not beyond the range of M ±w.

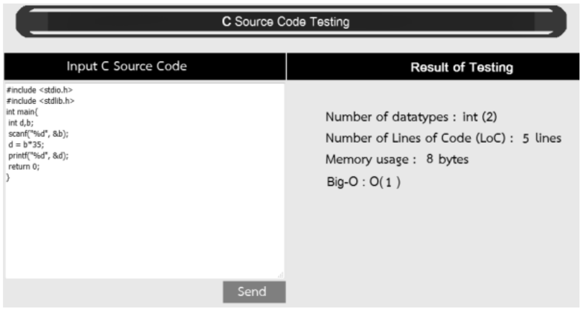
Meanwhile, the part of the student supposes a mean value of the memory usage is 66. Thus the performance level in memory usage is falling in the range of very good with 95% of CI. Also, if his/her mean value is 68, the evaluation of the performance of the memory usage would be in a good level with 95% of CI. On the other hand, the performance level is not in the standard based on the experts if the mean value is out of range.

2) ***Form Notation Measurement***

This function is scanning and checking the key word of loop commands such as "for", "while", and "do while". Since the target group of this research is the first/second year student of the university, our proposed tool provides only the simple notation of Big-O, which are O(1), O(n^2),...,O(n^c), where c is a constant integer.

IV. RESULTS

Our tool is developed by using C# language running on an IIS web server. Firstly, the subjects of a programming problem are sent into the system within one of two categories: basic and complex. For instance, a student sent their source code for solving the problem 1; the calculation of the money exchange from $US to Thai baht. After completely compiling without any error and given the correct answer, then the analyzer runs with a string scanning technique. The result shows that: 1) the number of data type has 2 integer, 2) the number of line of code has 5 lines, 3) the memory usage is 8 bytes, and 4) Big-O notation is O(1) as shown in Fig. 2.

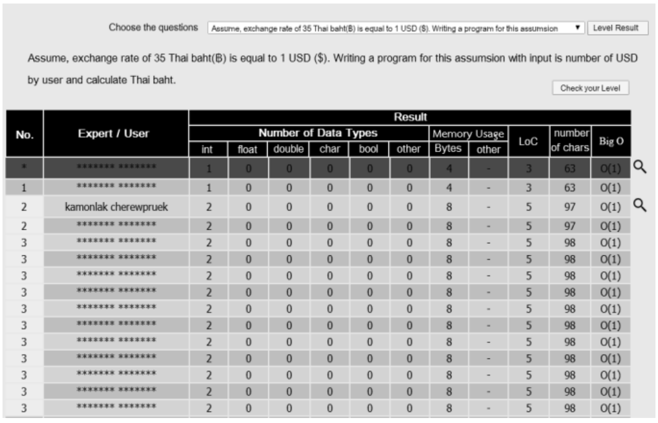


```
#include <stdio.h>
#include <stdlib.h>
int main{
 int d,b;
 scanf("%d", &b);
 d = b*35;
 printf("%d", &d);
 return 0;
}
```

Number of datatypes : int (2)
Number of Lines of Code (LoC) : 5 lines
Memory usage : 8 bytes
Big-O : O(1)

Fig. 2 Source Code and Results

Secondly, all results are shown in the ranking list according to the problem solving or the question. In this case, the student named "kamonlak" was ranked in third place. It is noted that other students' names are not disclosed, including the experts' name, which is on the first row with slightly dark background color. Each student can check their ranking and source code design comparing to the others, particularly regarding the expert with a magnifying glass icon as shown in Fig. 3.



Choose the questions: Assume, exchange rate of 35 Thai baht(฿) is equal to 1 USD ($). Writing a program for this assumsion

Assume, exchange rate of 35 Thai baht(฿) is equal to 1 USD ($). Writing a program for this assumsion with input is number of USD by user and calculate Thai baht.

No.	Expert / User	Result										
		Number of Data Types						Memory Usage		LoC	number of chars	Big O
		int	float	double	char	bool	other	Bytes	other			
*	******* *******	1	0	0	0	0	0	4	-	3	63	O(1)
1	******* *******	1	0	0	0	0	0	4	-	3	63	O(1)
2	kamonlak cherewpruek	2	0	0	0	0	0	8	-	5	97	O(1)
2	******* *******	2	0	0	0	0	0	8	-	5	97	O(1)
3	******* *******	2	0	0	0	0	0	8	-	5	98	O(1)
3	******* *******	2	0	0	0	0	0	8	-	5	98	O(1)
3	******* *******	2	0	0	0	0	0	8	-	5	98	O(1)
3	******* *******	2	0	0	0	0	0	8	-	5	98	O(1)
3	******* *******	2	0	0	0	0	0	8	-	5	98	O(1)
3	******* *******	2	0	0	0	0	0	8	-	5	98	O(1)
3	******* *******	2	0	0	0	0	0	8	-	5	98	O(1)
3	******* *******	2	0	0	0	0	0	8	-	5	98	O(1)
3	******* *******	2	0	0	0	0	0	8	-	5	98	O(1)
3	******* *******	2	0	0	0	0	0	8	-	5	98	O(1)
3	******* *******	2	0	0	0	0	0	8	-	5	98	O(1)

Fig. 3 Expert/User Results in Ranking List

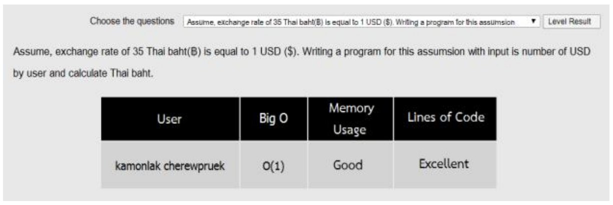
TABLE I
MEMORY USAGE MEASUREMENT

	Memory Usage					Mean	Confidence Interval	Student Mean	Performance Result
	Expert 1	Expert 2	Expert 3	Expert 4	Expert 5				
Problem 1	4	4	4	8	8	5.6	4 - 8	8	Good

TABLE II
LINES OF CODE MEASUREMENT

	Lines of Code (LoC)					Mean	Confidence Interval	Student Mean	Performance Result
	Expert 1	Expert 2	Expert 3	Expert 4	Expert 5				
Problem 1	3	5	5	7	8	5.6	4 - 7	5	Excellent

Finally, the memory usage and the line of code are evaluated as shown in Table I and Table II. In this case, there are 5 experts sending the source of the problem 1. In practice, each programmer is differently coding in different algorithms. It is hard to justify the problem solving which is an open question. So our tool based on the experts, measures the students' skill and performance. The overall result of student's skill and performance are shown in Fig. 4.


Choose the questions: Assume, exchange rate of 35 Thai baht(฿) is equal to 1 USD ($). Writing a program for this assumsion ▼ | Level Result

Assume, exchange rate of 35 Thai baht(฿) is equal to 1 USD ($). Writing a program for this assumsion with input is number of USD by user and calculate Thai baht.

User	Big O	Memory Usage	Lines of Code
kamonlak cherewpruek	O(1)	Good	Excellent

Fig. 4 Evaluation Results Screen

V. DISCUSSION AND CONCLUSION

There is a lot of research proposing some techniques for expanding efficiency on a compiler [7] such as reducing ambiguity of Res [1] and creating a pattern-matching tool for increasing the efficiency of compiler implementation [2-4]. In addition, source code performance is concerned with string scanning techniques such as a single keyword pattern matching technique [5]. This research proposes an idea for increasing the potential of a tool (compiler) in system performance. In addition, the potential of input should be concerned such as source code. Especially, if a source code is created in an appropriate design then it will bring good system performance.

This research proposed an enhanced tool for improving source code and could be an optimized source code for problem solving. The proposed solution in the designed core module can be categorized into 3 mechanisms. In the first section, the source code is analyzed to identify memory usage from the number of each data type. The number of lines of code is proposed in the second section. In the third, the analysis of the basic Big-O notation is studied on the pattern of source code design. Moreover, source code from the experts will be analyzed as the same 3 mechanisms in each programming problem, so called the Expert Base Analysis (EBA) mechanism. Therefore, a user can use an EBA mechanism for measuring programming skill. The advantage of this research is to help students, such as programming students in computer science, analyze their algorithm design for adapting their skill appropriately.

VI. ACKNOWLEDGEMENT

This research was supported by a grant from Nakhon Pathom Rajabhat University (NPRU. The findings in this article are those of the authors and do not necessary reflect the views of the NPRU.

REFERENCES

(Arranged in the order of citation in the same fashion as the case of Footnotes.)

[1] Paul, S. and Prakash, A. (1994). "A framework for source code search using program patterns". IEEE Transactions on Software Engineering, Vol. 20(6), pp. 463-475.

[2] Kuhnemann, M., Rauber, T., and Runger, G. (2004). "A source code analyzer for performance prediction". 18th International Parallel and Distributed Processing Symposium, April 26-30, 2004.

[3] Adve, V.S., Mellor-Crummey, J., and Anderson, M. (1995). "An Integrated Compilation and Performance Analysis Environment for Data Parallel Programs". IEEE/ACM SC95 Conference.

[4] Ford, B. (2004). "Parsing expression grammars: a recognition-based syntactic foundation". 31st ACM SIGPLAN-SIGACT symposium on Principles of programming languages, January 14-16, 2004, Venice, Italy, pp. 111-122.

[5] Ierusalimschy, R. (2009). "A text pattern-matching tool based on Parsing Expression Grammars". Journal of Software-Practice & Experience, Vol. 39(3), pp. 221-258.

[6] Jeffery, C., Thomas, P., Gaikaiwari, S., and Goettsche, J. (2016). "Integrating

regular expressions and SNOBOL patterns into string scanning: a unifying approach". 31st Annual ACM Symposium on Applied Computing, April 4-8, 2016, Pisa, Italy, pp. 1974-1979.

[7] Kourie, D.G., Watson, B.W., Strauss, T., Cleophas, L., and Mauch, M. (2014). "Empirically Assessing Algorithm Performance". the Southern African Institute for Computer Scientist and Information Technologists Annual Conference, September 29 - October 1, 2014, Centurion, South Africa, pp. 115.

[8] Dang, D.C. and Lehre, P.K. (2015). "Simplified Runtime Analysis of Estimation of Distribution Algorithms". Annual Conference on Genetic and Evolutionary Computation, July 11-15, 2015, Madrid, Spain, pp. 513-518.

[9] Terai, M., Murai, H., and Minami, K. (2012). "K-scope: A Java-Based Fortran Source Code Analyzer with Graphical User Interface for Performance Improvement". 41st International Conference on Parallel Processing Workshops (ICPPW), September 10-13, 2012.