JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 38, 275-294 (2022)
DOI: 10.6688/JISE.202203_38(2).0001

Online Judgment System for Assessing C Program Structure*

CHORNG-SHIUH KOONG, ZIH-YU WEI, CHUN-HSIEN CHEN AND CHENG-YAN GUO+
Department of Computer Science
National Taichung University of Education
Taichung, 403 Taiwan
E-mail: csko@mail.ntcu.edu.tw; {bcs106119; bcs108110}@gm.ntcu.edu.tw; R04458006@ntu.edu.tw

The Online Judge (OJ) system solves many teaching problems of programming course design in traditional teaching modes and enhances teaching quality and learning effectiveness. Most of the OJ systems rate students by compiling and executing their code and then comparing the output with the standard output. However, program code content and students' ability to program structure cannot be accessed with such design. To improve this shortcoming, we have developed an "automatic structure assessment module" and added it to the existing system. The module functions to check the student program structure, enabling the OJ system to measure code quality. Such extended functionalities enable the OJ system to produce more accurate assessments of programming learning performance. Furthermore, how the program structure can be described in the program structure specification is another challenge. Thus, we have also developed a visual program structure description editor. The teacher can handily specify the program structure specification required by the program structure according to the teaching topic so that the OJ system can execute program structure checking. The functionalities of automatic program structure assessment combined with the visual program structure description editor can be used to rapidly generate many examinations and practice questions that require program structure evaluation, and thus truly achieve the efficiency of automatic program structure assessment as well as enhancing students' quality of program structure. Finally, in the six-week teaching experiment, there are a total of 61 valid samples. Using our structure comparison system can effectively improve the learning effectiveness of low-score group students in programming.

Keywords: program structure assessment, automatic assessment, online judge system, programming teaching, syntax tree, programming language

1. INTRODUCTION

C language is a widely used computer language and also an introductory course in many related departments of computer science of universities [1, 2]. However, it is not easy to learn well for first-year students. For programming courses that involve practices, traditional teaching modes are not sufficient. Learning setbacks often arise in the learning process [3]. Compared with simple lecture teaching modes, exercises are more effective in learning programming [4]. However, it is difficult for teachers to check the students' programming code one by one in class, and thus they find it hard to assess student learning [5, 6]. Therefore, many teachers use the "Online Judge (OJ) system" to facilitate teaching,

Received March 11, 2020; revised November 6, 2020 & January 3, 2021; accepted February 5, 2021.
Communicated by Tzung-Pei Hong.
+ Corresponding author.
* This study is supported in part by the Ministry of Science and Technology in Taiwan under contract numbers MOST107-2511-H-142-008-MY2, 109-2511-H-142-008, and 109-2511-H-142-010-MY3.

allowing learners to quickly receive feedback and teachers to track student learning in real-time. The OJ system has many benefits. It offers abundant opportunities for hands-on practices, improves the efficiency of scoring, provides instant feedback, and establishes objective assessment criteria [7-12]. However, it also has shortcomings. The general OJ system only checks the executing output, not the quality of the program [7]. Therefore, it cannot judge whether the program structure is written according to the required structure of test questions [11]. When taking exercises, some students focus on how to pass the test cases rather than how to use the new concepts of program structure. By using OJ system, teachers cannot make sure whether their students learn new required programming skills [9]. It makes such OJ systems less helpful to improve programming ability and cultivate the knowledge of program structure. Program structure knowledge is critical because it is the foundation of many required courses, such as data structures, algorithms, and so on. Data structure and algorithm are the core abilities of information engineering in computer science [13]. Learning how to write codes and algorithm are complementary [14]. The data structure is also the introductory course of the Department of Information Science [15]. These methods and principles are the basis for learning compilation, design, system operation, and database [16]. Therefore, without a conception of program structure, students will encounter many difficulties when taking these required courses. To solve this problem, we have developed the automatic program structure assessment module to enhance our original OJ system, enabling it not only to check the program output, but also the program structure.

The main concept is to convert the student's code into a syntax tree, and then compare the student code's syntax tree with the required template syntax tree. This research uses combinatorics expressions to describe our research methods, define the descriptive syntax tree rules of program structure representation, and develop program structure comparison algorithms. Also, the program structure description is more complicated to express in text format, it is difficult to create a large number of program questions quickly. To assist teachers to quickly create program structure description, we also have developed the visual editor. The visual editor can allow teachers visually to create program structure requirements for examination questions.

2. RELATED WORK

2.1 Online Judge System

OJ systems have been used for many years [7, 17]. Currently, they are more widely applied in programming courses. Such systems can immediately judge source code from the students automatically [7]. The traditional OJ system cannot meet the practice needs of teaching; so many researchers develop extended functionalities for the OJ system on demand. The following will introduce the extended functionalities and related works.

Higgins *et al.* developed an OJ system with JAVA named Course Marker and applied it to improve the lack of network connectivity in the old system Ceilidh, and enhanced the system's capability of feedback [18]. The system supports the scoring of Java and C++ and conducts testing by compiling the code and comparing the test data [18]. Following the ACM Programming Contest model, Jianhua Wu *et al.* developed an online OJ system to assist students in program coding. The system supports Assembler, C/C++, Pascal, Java, C#, *etc.* It compiles, executes the code, analyzes the results, and then per-forms evaluations,

hence improving its programming ability considerably [12]. Hui Sun *et al.* designed the YOJ system as a teaching assisted tool for programming courses. They considered that the available systems only pay attention to whether problems are solved in a given time, rather than how. Consequently, they developed an answering mode of filling in the blank. Students need to fill in some code, and the YOJ system only can check how students understand specific knowledge [8].

Siwan Tang, *et al.* Studied and analyzed the MOOC's automatic scoring system tools (Massive Open Online Course). They found that programming capability can be improved with the online judge system. On the other hand, it can reduce the workload of teachers. However, they found some shortcomings in long-term use: only considering the output without checking the content, the system cannot distinguish how serious the error codes. Therefore, they have proposed a model of keyword syntax check in the system to make up for the shortcomings [9].

Aizu University Database System Laboratory analyzed the recorded data uploaded by beginners to Aizuo OJ System. The results showed that many programmers hope to identify defects in the code at an early stage of writing the program to reduce the risk of errors [19]. Zhou, *et al.* [20] raise some doubts about the traditional OJ system:

- Rating the program based on the number of test cases passed only, rather than comparing code quality.
- Lack of personalized feedback. When a student code is incorrect, they can get the error information of test cases only, unable to know the exact reason. Sometimes a small error will take much time to fix, which brings great trouble to the course.
- The code plagiarism problem. The existing OJ system does not pay attention to plagiarism detection.

Zhou *et al.* have designed a new OJ system to solve these problems. They provide code similarity checking, and code quality evaluation and personalized feedback analysis tools are still under development [20].

Poon *et al.* thought that there are many forms of programming problems. Thus, it is not appropriate if the OJ system can only support the output's accurate matching answer. The system may judge as an error a small difference between the output and test case due to a minor difference after the decimal point, which may discourage learners and even make them lose their motivation to learn. At the same time, teachers feel restricted in the flexibility of test design. To resolve this problem, they proposed the Hierarchical Program Output Structure (HiPOS) system, which provides more expressiveness and flexibility to the output and can build modeling of the program output. They conducted experiments and computed the average grades, and they compared the automatic evaluation method and manual judgment approach, which got the results of accuracy of 0.8467 [21]. It is difficult for novices to master the syntax of programming languages. Providing enhanced compiler errors may not help them fix syntax errors. Therefore, more evidence is required that the professionally-oriented static analysis tool does identify factual errors in beginners' code [22-24].

The above OJ systems evaluate programs mainly by inputting test cases and comparing output; some will improve their shortcomings. Such as scoring specific knowledge, keyword verification, code quality evaluation, personalized feedback, and code similarity checks, *etc.* For example, the YOJ system of Hui Sun *et al.* uses the mode of filling-in-the-

blank questions to check whether students can learn specific knowledge. Although it is helpful for students to learn, we hope students can further obtain the program execution results and code quality. Such as information about structure detection, while practicing and submitting the code, Siwan, Tang *et al.* designed a system capable of checking the program content with keyword verification.

However, we prefer to look into the program's whole structure since keyword verification can find what we want, but it cannot tell us where the error lies. For example, if we want students to use nested for-loop, we hope to check whether they place one loop inside another's body, rather than just two for-loops. The conception of Zhou, *et al.* [20] is different from ours, but the code plagiarism detection is also worthy of further development.

2.2 Comparison of Current OJ System

Table 1 list the characteristics and research directions of the existing online judge system. Including ours and other related research in recent years, these systems can mainly support C/C++ assessment, and some systems such as that of Zhou, *et al.* [20] and our system can support JAVA language. Zhou, *et al.* [20] can also support some interpreted language such as Python and JavaScript. The judge mode can support general judgment, that is, the commonly used judge method of compiling and executing, then comparing the output results. The current studies of OJ systems are also based on this method and then develop their features. Some systems allow deviation in comparing output results (such as floating-point numbers) in comparing output results, making the answer more flexible. Poon, *et al.* [21] and our system have the function of allowing the range of floating-point deviation. Our system also provides a comparison mode for ignoring symbols like spaces and line breaks for string comparison. In order to overcome the shortcomings of OJ system that only checks program execution results, the current trend of OJ system has been shifted to focus on checking program content and structure. Sun and Tang respectively proposed static detection methods for filling in questions and keywords detection. Zhou, *et al.* [20] proposed to use Machine Learning to analyze the code and provide personalized feedback. Pham and Nguyen [25] research is to design a solution to prevent plagiarism. The research of Kasahara, *et al.* [26] is based on calculating the value of (CC) Cyclic Complexity and making the leaderboard of best CC value so that students would think about how to write a better code against each other to achieve the effect of improving students' programming skill indirectly. The development goals of Sun and Tang are similar to ours. However, we hope students can submit their completed code for scoring and get feedback on program execution results and program quality (such as program structure). Our purpose is to develop an automatic program structure assessment system to automatically detect the code's content by comparing whether the student's program has the expected structure. Personalized feedback is the goal of the Machine Learning of Zhou, *et al.* [20]. However, it is still uncertain how detailed the personalized feedback information currently available on their system. For achieving the effect of personalized feedback, our system can point out what structure is expected and where it should be when the students' program structure is wrong. As for the code quality assessment, Kasahara, *et al.* [26] uses calculating the program CC (Cyclic Complexity) to score the quality of the code. We think this is a straightforward concept, but a very innovative application method. Since our program structure analysis can also quickly analyze the Cyclic Complexity of the code, we may also consider following Kasahara, *et al.* [26] way to score the code quality in the future.

Table 1. Comparison of current OJ system.

	Sun (2014)	Tang (2016)	Poon (2018)	Zhou (2018)	Pham (2019)	Kasahara (2019)	Our study (2020)
Supported languages	C	No system	C++	C/C++ JAVA Python JavaScript	C++	C	C JAVA
Results assessment	V	No system	V	V	V	V	V
Fuzzy results assessment	X	No system	V	X	X	X	V
Source code content checking	Fill in code sentence	Keyword detection	X	Analysis by machine learning	Clone detecting	Calculate the value of Cyclic Complexity	Program structure assessment
Clone detecting	X	X	X	V	V	X	X
Personalized feedback	X	X	X	V	X	X	V
Code quality assessment	X	X	X	X	X	V	X

'V' denotes to have functionality; 'X' denotes to have not functionality.

3. SYSTEM DESIGN

Automatic program structure assessment is more complicated than traditional scoring of program output. Its primary focus is on how to describe the structure of the code and how to allow OJ-System to perform automatic assessments accordingly. To perform an automatic structure assessment, we design a process for creating a test question, and a visual program structure editing tool is used. The required code structure is edited visually, and then program structure specification is generated, which is used as test cases. Then, through the expanded program modules of the OJ-System, automatic structural assessment is conducted. The available automatic scoring system requires a unified output format specification before scoring. In this study, the expected program structure is converted into a specific expression format, and so is the student's code. The student's code is compared with the expected program structure in this format, and automatic structural inspection is conducted. The Syntax tree of the program code is a specific unified format. As shown in the example in Fig. 1, if we can convert the student's code into a syntax tree, after simplification, it is compared with our standardized structure grammar. The system can detect whether the student's program structure conforms to the question specification.

3.1 The Principles of Structural Inspection

The concept of student program structure comparison is mainly to convert the student's code into a syntax tree, and then compare the student code's syntax tree with the template pattern syntax tree required by the question to validate whether the student's program structure meets the requirements of the question (Fig. 1). It is used to improve the quality of student code when implementing programming teaching. We will explain in more detail in this chapter.

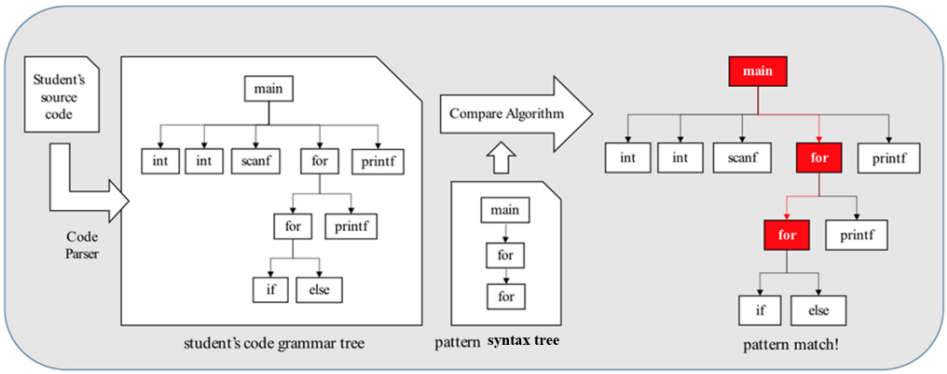


Fig. 1. The principles of structural inspection.

We define the similarity of two programs using Eq. (1). *S* and *T* are the abstract syntax tree (AST) token set of the student program and the required template structure program. Where *Sim*(*S*, *T*) can be expressed as the degree of intersection of *S* and *T*.

$$Sim(S,T) \equiv \frac{|\{S_i|(s_i,t_j)\in R_s\}|+|\{T_i|(s_i,t_j)\in R_s\}|}{|S|+|T|}, 0 \le Sim(S,T) \le 1 \tag{1}$$

In Fig. 2, *R* is the AST token matching of *S* and *T*. The set is defined as $R \subset S \times T$, and when $T \subseteq S$, it means that there are indeed some grammatical similarities in the student program [27].

Fig. 2. Correspondence of elements *R*.

The intersection between the AST token sets of the software system can only indicate the similarity of the two software systems' grammatical use. It cannot quantify the degree of structural similarity between the software systems. For the degree of structural similarity, the Eq. (2) is a subtree matching algorithm can detect the maximum length matching of the AST subtree that meets the template program in the student program AST. The tiles are the set of matching intersections of all subtrees of AST of *S* and *T* [28]. The maximal_matching is the maximum length of subtree matching in [28], where tiles are the set of matching of all subtrees Jiang, *et al.* [29]. When the maximal_matching(*S*, *T*) of the AST intersection of *S* and *T* is 1, then $S = T$, $0 \le \text{maximal\_matching}(S, T) \le 1$.

$$maximal\_matching(S,T) = \frac{2\times tiles}{|S|+|T|}, 0 \le maximal\_matching(S,T) \le 1$$
$$tiles = \sum_{subtree\_matching(s,t,length)\in tiles} length \tag{2}$$

If Eq. (3) are equal, the student program AST has all the elements in the specified template structure program AST, and the degree of structural similarity is precisely the same. If unequal indicates that the maximum length of the AST subtree matches, and it is not equal to the size of all tokens in the template structure program AST, the structure is considered to be different.

$$\begin{cases} maximal\_matching(S,T) = \frac{|T|}{|S|+|T|}, T \text{ is a subtree of } S \\ maximal\_matching(S,T) = \frac{|T|}{|S|+|T|}, T \text{ is not a subtree of } S \end{cases} \tag{3}$$

3.2 Comparison Algorithm for Program Structure

To achieve automatic structure scoring and keep coding flexibility, we use parser Bendersky [30] to convert the program into a syntax tree. Then, through the function of tree reduction developed, it is converted into an intermediate format defined by our system. The required structure representation is also expressed in our defined intermediate format, Table 2 is used to indicate the node of the rule; each node can be divided into three parts: level of a node, type, and secondary attributes.

Table 2. Structural rule keyword table.

Types	Description	example
ArrayDecl	Declare array	int s[100];
Assignment	Assign variable	i=0;
BinaryOp	Operator，ex.>,<,==,>=,!=, *etc.*	x<100;
Break	Break statement	if(a>b) break;
Case	Case statement in switch	case 'A':
Compound	Code block{}	int main(){ … }
Constant	Constant or string	a = 100;
Continue	continue statement	if(a<b) continue;
Decl	Declare variables	int a;
DeclList	Declare variables in the for loop	for(int i=0; …)
Default	Default statement in switch	default: …
DoWhile	do while statement	do{...}while(…)
For	for statement	for(int i=0;i<=10;i++)
FuncCall	Call function	printf("Hello world!");
FuncDecl	Define function	int sum(int a, int b){...}
ID	Use variable	printf("%d",sum);
IdentifierType	Variable type	int i;
If	if statement	if(a>b) {…};
Return	return statement	return 0;
Struct	struct statement	struct person{…}
Switch	switch statement	switch(a){…}
TypeDecl	Declare variable, use with Identi-	int i;
UnaryOp	Unary operators, ex.&,,--,*etc.*	scanf("%d",&a); i++;
While	while statement	while(…){…}

According to the combinatorics in the previous section, in order to detect whether the student's program meets the requirements of the program structure of the test question, we have designed an algorithm for structure comparison, which mainly uses recursive methods to compare the nodes of the syntax tree, as shown below in Algorithm I. The pseudo-code is shown below. Each grammar node has several significant attributes for comparison. It contains:

- **mainValue (primary attribute):** to store the main identity information of the node, such as Binary Operation, IdentifierType, for, while, *etc.*
- **subValue (secondary attribute):** to store secondary data of the node, such as Binary Operation may be the comparison symbol >=, IdentifierType may be int type *etc.*
- **Parent node:** to record the upper node of the node.
- **Child node:** to record which lower nodes the node has.
- **nodeCompare(A, B):** Compare data between A and B, when the mainValue and subValue of the two nodes are the same, return true; otherwise, return false. When B's mainValue is "ANY," any data of the A node can be matched must return true. When B's mainValue is "NOT," it means that the mainValue of A is not in the subValue of B. If it does not have it, it will return true; otherwise, it will return false.
- **setTarget(targetNode):** When a node that does not meet the expectations is detected, this function will store the node in a global variable targetNode. The subtree matching algorithm will determine the unmatched node after the execution.

Algorithm I: Structure Compare

```
Input:
nodeA ← node of the student code tree.
nodeB ← node of the except code tree.
FUNCTION treeComparator(nodeA,nodeB) :
    global targetNode
    R ← 'init'
    IF len(nodeB.children) > 0 :
        index ← 0
        FOR B in nodeB.children :
            IF len(nodeA.children) > index :
                FOR A in nodeA.children[index:] :
                    index += 1
                    IF nodeCompare(A,B):
                        R ← treeComparator(A,B)
                        IF R is True:
                            targetNode ← None
                            break
                        ELSEIF R is False:
                            setTarget(targetNode)
                            RETURN False
                        ELSE:   # R ← None
                            continue
                        ENDIF
                    ELSEIF B.mainValue = "NOT" AND
                            (A.mainValue in B.subValue OR B.subValue in A.mainValue):
```

```
                        IF setTarget(targetNode)
                        RETURN False
                    ELSEIF index = len(nodeA.children):
                        setTarget(targetNode)
                        RETURN None
                    ENDIF
                ENDFOR
            ELSE:
                setTarget(targetNode)
                RETURN None
            ENDIF
        ENDFOR
    ENDIF
    IF R is None AND nodeB.mainValue = 'root':
        RETURN False
    ELSEIF R is None :
        RETURN None
    ELSE:
        RETURN True
    ENDIF
```

For an AST of N nodes, this comparison process is $\boldsymbol{O(N^3)}$, and empirically, an extensive software system of M lines of code has $\boldsymbol{N = 10*M}$ AST nodes [31].

3.3 Inspection Example Demonstration

We illustrate with practical examples. The test questions require students to practice using the double-layer for loop to print the nine-nine multiplication table. As shown in Figs. 3 and 4, it is a practical case of the correct student answer. It is implemented with a double-layer for-loop, and it passes the automatic structure inspection. As shown in Figs. 5 and 6, it is an actual case of the incorrect student answer. It is implemented with one layer of for-loop and adds the printf() instruction. The automatic structure scoring system will check for errors, as shown in Fig. 6. The general OJ system cannot detect such errors.

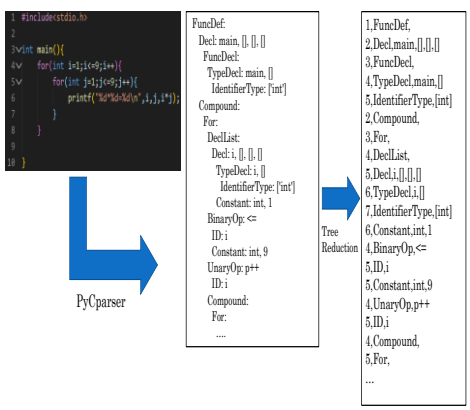


Fig. 3. Correctly structured code.

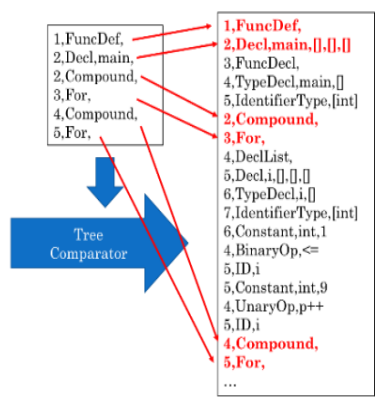


Fig. 4. Code with correct structure will pass the inspection.

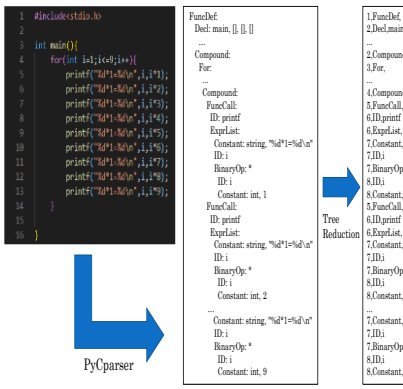


Fig. 5. Code with incorrect structure.

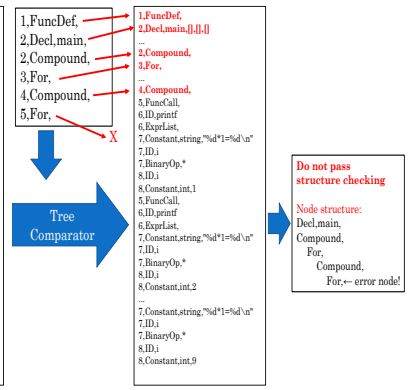


Fig. 6. Code with incorrect structure will not pass the inspection.

3.4 Visual Editor for Program Structure Description

The structure rule statement has a specific format, and the required program structure description file must be writhed in this format before a comparison can be performed. In order to allow teachers to edit the desired structure rules conveniently and quickly, we developed a visual structure rule editor, which can be used in the visual syntax tree mode or the programming mode to write structure rules. The visual syntax tree editor is shown in Fig. 7. The left is the edit area, and the right is the structure rule preview area; the programming mode and visual syntax tree mode can be switched, the structure syntax tree can be previewed, and the new/add/delete node functions are provided. The current visual syntax tree model can support the generation of grammar nodes related to process control.

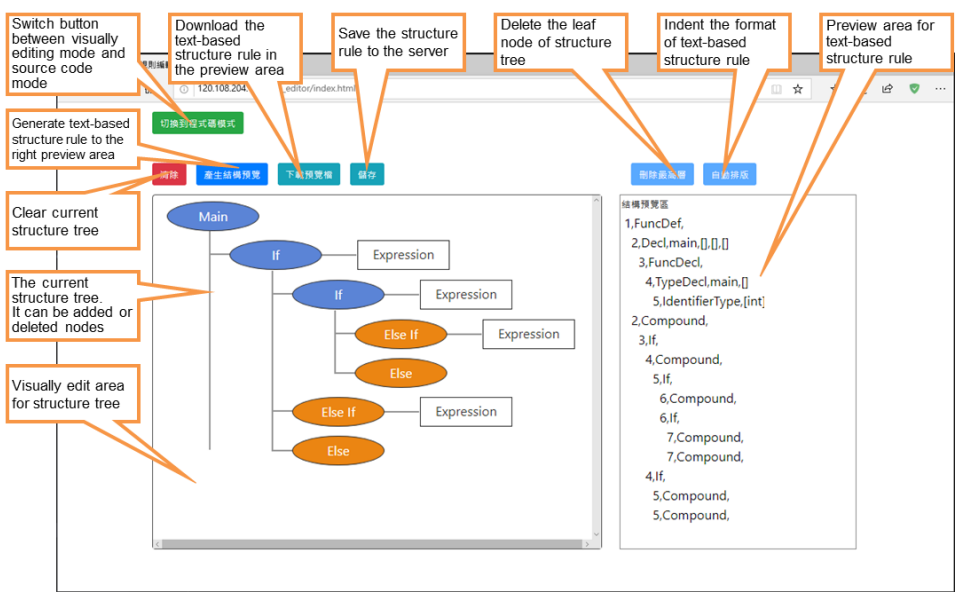


Fig. 7. Visual editor for program structure description.

4. SYSTEM DEMONSTRATION AND EXAMPLE

4.1 The Assessment Process

We have embedded the automatic structure assessment module into the previously developed OJ System to strengthen the functionalities. We have also developed a user interface for teachers to write structural answers. The following are the introduction to our system's assessment process and the functional description of the module.

The automatic assessment process is shown in Fig. 8. When the student submits the code once, the system will perform test case comparison and structure comparison if the compilation is successful. The test case comparison will calculate the score according to the number of cases passed. The source code will be converted into a structure tree for structure comparison and compared with the "expected structure" tree to detect its expected structure. When the structure is correct, we will report the message that the code passes the structure check, but if the structure is wrong, we will find out what the error is and output an analysis report to the student.

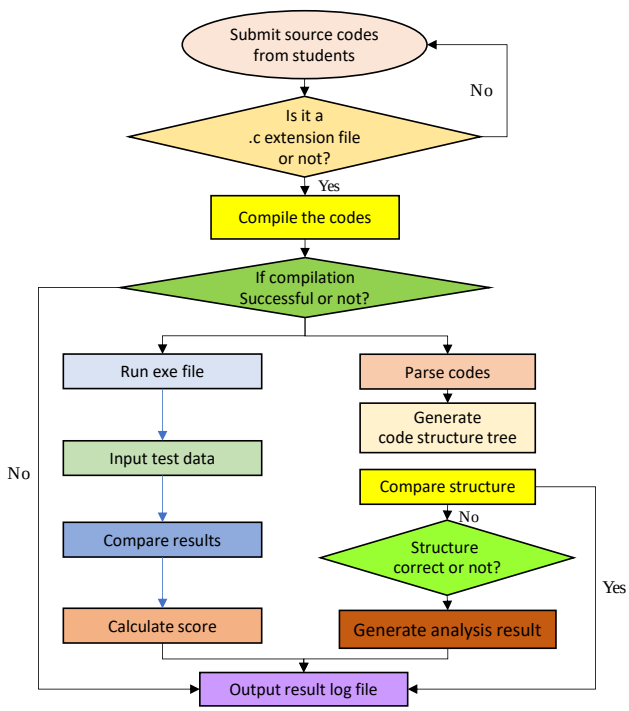


Fig. 8. The process of structural assessment.

4.2 The System Modules of Automatic Structure Assessment

Automatic structure assessment module is mainly composed of the Structure Assessment Module and Visual Structure Assessment Module (Fig. 9). The two modules are described as follows:

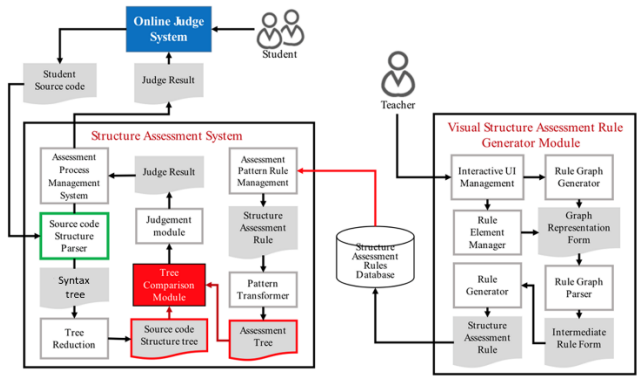


Fig. 9. Automatic structure assessment module.

(A) Structure Assessment Module

The primary function is to compare students' source code captured by the automatic assessment system and the required structure to check whether the structure is correct and return the result to the automatic assessment system. "Assessment Process Management Module" is responsible for controlling the incoming data and the output of results. The incoming code will first be parsed into "Syntax tree" and via "Tree Reduction Module" the parts we need are analyzed, and the "Student source code Structure tree" is generated. "Assessment Pattern Rule Management Module" will find the "Structure Assessment Rule" of the test question from the database and convert it to "Assessment Tree" for comparison through the "Pattern Transformer Module". "Assessment Tree" and "Student source code Structure tree" are sent to and compared in "Tree Comparison module". Thus, the entered code structure tree can be checked, whether it contains the expected tree structure. And the comparison result will be passed to "Judgment Module" to judge whether the tested code passes the structure comparison. With "Assessment Process Management Module", the information will be sent back to the OJ system for users to view the inspection results.

(B) Visual Structure Assessment Rule Generator Module

To execute structural comparison, OJ systems must work with Structure Assessment Rule. However, it is more challenging to generate Structure Assessment Rule than traditional test data. To allow teachers to design "Assessment Rule" quickly and conveniently, we have also developed an online user interface, "Visual Structure Assessment Rule Generator Module". With an interactive menu, teachers can easily create Assessment Rule, which is icon-based. Fig. 9 shows the major function of "Interactive UI Management Module" is to manage web components that provide teachers with an interactive visual user interface. "Rule Element Manager Module" is mainly to create and manage Assessment Rule components, such as if, while, for, *etc.* "Rule Graph Generator Module" is in charge of the relations among the components of "Assessment Rule", allowing several components to compose "Visual Structure Tree" that is "Assessment Rule" itself. The "Interactive UI Management Module" controls "Rule Element Manager Module" and "Rule Graph Generator Module" simultaneously so that the teachers can design and set the structure rules. When "Visual Structure Tree" is edited by the teachers, intermediate format data "Graph Representation Form" is created. "Graph Representation Form" will be converted,

and via "Rule Graph Module" and "Rule Generator", text-based "Structure Assessment Module" will be generated and stored in the database for later evaluation.

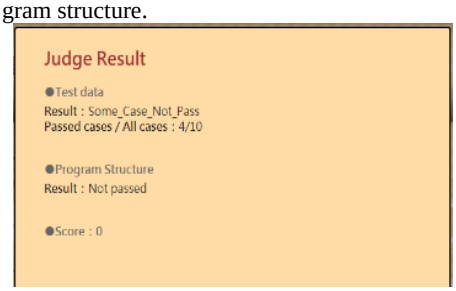
4.3 System Demonstration

The following is the screen display when our system is executed. Here are the explanations:

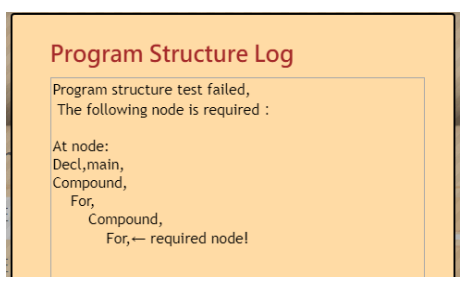
Step 1: On the answer page, the questions will be displayed here, and students can answer them here, view the previous detailed answer records *etc.* in Fig. 10 (a).
Step 2: Online writing mode: students can write code directly on the webpage, and assessment is conducted. In addition, the code can be submitted in the mode of uploading files in Fig. 10 (b).
Step 3: After the code is submitted, the system will display the results of the evaluation of test cases and structural inspection in Fig. 10 (c).
Step 4: If the structure check is not passed, the system will inform students of the result of non-conformity. The example is that a double-layer for-loop needs to be written, but only a single layer is detected in Fig. 10 (d).
Step 5: The teachers can use the visual structure description editor to specify the required program structure specification, and add or delete the required program structure by clicking on the component in Fig. 10 (e).
Step 6: The picture shows an example of editing. When the editing is completed, the system will generate a syntax tree on the right, which is used to check a test question's structure in Fig. 10 (f).

(a) The screenshot of the test question for program structure.

(b) The screenshot of the student's answer.

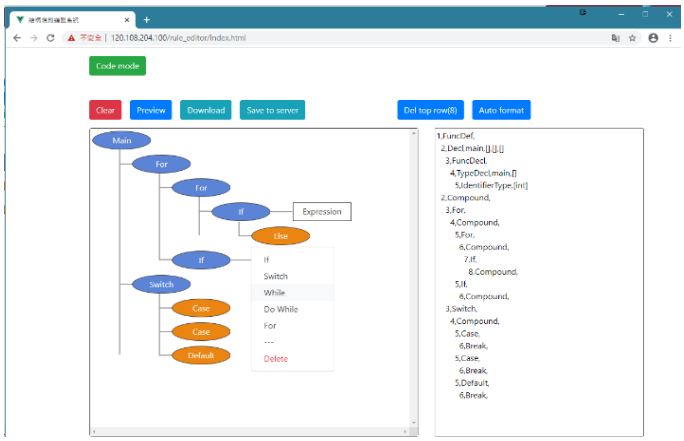


(c) The screenshot of the automatic assessment result.

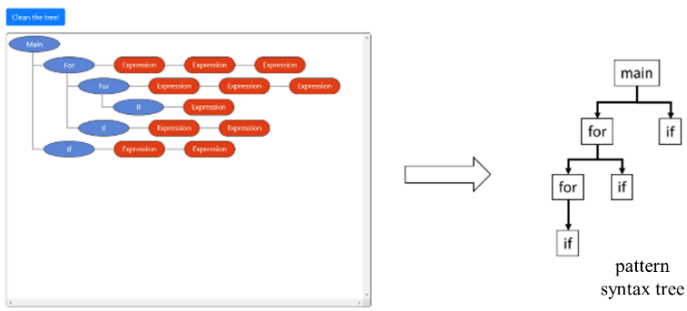


(d) The screenshot of the structural inspection result.

Fig. 10. The screenshot of system demonstration.



(e) Visual program structure description editor.



(f) Syntax tree pattern generated from system.

Fig. 10. (Cont'd) The screenshot of system demonstration.

5. EXPERIMENT AND DISCUSSION

This experiment examines whether the students' concept of program structure improved after the automatic structure comparison system was added to the OJ system. The experiment's design shown in Fig. 12, including the control variable, includes the teacher, teaching materials, and exercise questions; the independent variable is whether there is a structure comparison system; the dependent variable is the students' learning effectiveness.

The experiment process is as Fig. 11. We first teach the first-year student in the C-language program structure for three weeks and then conduct the pre-test after students practice. There are 19 questions in total, and s-type grouping into two groups according to the pre-test scores from high to low. Then a two weeks review was conducted. The experimental group used an OJ system with structural inspection, and the control group used an OJ system without a structural inspection. The difficulty of the content is the same, and experts have checked the questions.

The experimental results are 61 valid samples, 32 valid samples in the control group, 29 valid samples in the experimental group. According to the improved scores between the pre-test and post-tests, the whole independence sample *T*-test showed no significant difference between the two groups.

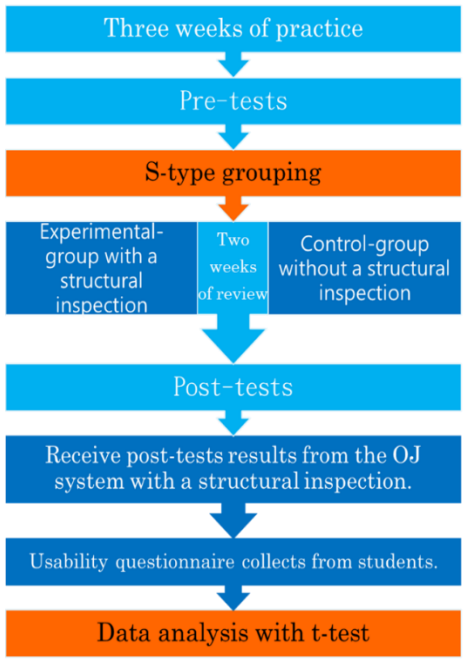


Fig. 11. Experiment flow chart.

Further analysis, the control group and the experimental group were divided into high-score group (top 33%), middle-score group (middle 34%), and low-score group (bottom 33%) according to the pre-test results.

Based on the improved scores to do an independent sample *T*-test, there is no significant difference between the control and experimental groups for the high-score and middle-score groups. It can be seen that the system has less effect on improving the learning effectiveness of students with high-score group and middle-score group learning achievements.

However, for the low-score group 11 students in the control group, 10 students in the experimental group. Independent sample *T*-test, the *p*-value is 0.0213*, indicating that the two groups' average improved scores are significantly different. The control group has an average improvement of 2.0909 questions and a standard deviation of 2.1659. The experimental group made an average improvement of 4.1 questions, with a standard deviation of 1.3703, as shown in Table 3. The experimental group's significance is greater than that of the control group, which means that the program structure inspection system can improve the learning effectiveness of low-group groups after the program structure inspection system is used for students' review.

This result is similar to Wu, *et al.* [11]. Low-score group students have poor basic knowledge and autonomy. Therefore, when they do not use structure testing, they may use standard methods to solve problems while avoiding learning new knowledge and concepts. Using structure comparison will allow students to write according to the topic's structural requirements and then promote them to learn new knowledge.

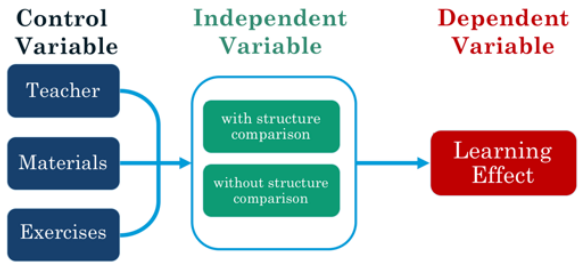


Fig. 12. Experimental design.

Table 3. The *t*-test result of learning effect for high, middle and low score group.

Group	*n*	Mean	SD	*p*-value
High				
Control	11	0.2727	1.4206	0.4793
Experimental	10	0.1	1.792	
Median				
Control	10	2.2000	1.8135	0.4952
Experimental	9	1.4444	2.2973	
Low				
Control	11	2.0909	2.1659	**0.0213***
Experimental	10	4.1	1.3703	

*: $p < 0.05$, **: $p < 0.01$, ***: $p < 0.001$

6. CONCLUSIONS

The general OJ system only checks the executing output, not the quality of the program structure. To ensure the basic requirements of software structure quality and to solve the problem that the OJ system cannot judge the program structure, we have developed an automatic structure assessment module to analyze the program submitted by students. This module can also highlight where the student program does not conform to the required program structure. In our research uses combinatorics expressions to describe our research methods, define the descriptive grammar rules of program structure representation, and develop program structure comparison algorithms. We have also developed the visual editor for program structure description, allowing teachers to visually specify the program structure specification of the expected program structure so that the OJ system can automatically perform structural inspections.

As a significant advantage of our system, such a design makes it handy to set test questions and save the setting time, enabling the instructors to quickly create more test questions. Now, we have made this OJ System available online. Finally, in the six-week teaching experiment of programming course, the result shows that using our structure comparison system can effectively improve the learning effectiveness of low-score group students in programming.

In the future, we will conduct a large number of experiments on program structure as a reference for improving programming teaching, and we will further expand the application to compare more programming languages.

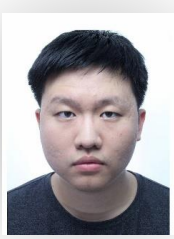
REFERENCES

1. D. Budny, L. Lund, J. Vipperman, and J. Patzer, "Four steps to teaching C programming," in *Proceedings of IEEE 32nd Annual Frontiers in Education*, 2002, Vol. 2, pp. F1G-F1G.
2. J.-B. Xie, "Problems in teaching of C programming language and their solutions [J]," *Journal of Chongqing University of Posts and Telecommunications* (*Social Science Edition*), Vol. 2, 2008, pp. 137-140.
3. A. J. Gomes and A. J. Mendes, "A study on student performance in first year CS courses," in *Proceedings of the 15th ACM Annual Conference on Innovation and Technology in Computer Science Education*, 2010, pp. 113-117.
4. Á. Matthíasdóttir, "How to teach programming languages to novice students? Lecturing or not," in *Proceedings of International Conference on Computer Systems and Technologies-CompSysTech*, Vol. 6, 2006, pp. 15-16.
5. H. Gao, Z. Qiu, D. Wu, and L. Gao, "Research and reflection on teaching of C programming language design," in *Proceedings of International Conference of Young Computer Scientists, Engineers and Educators*, 2015, pp. 370-377.
6. C. A. Higgins, G. Gray, P. Symeonidis, and A. Tsintsifas, "Automated assessment and experiences of teaching programming," *Journal on Educational Resources in Computing*, Vol. 5, 2005, p. 5.
7. A. Kurnia, A. Lim, and B. Cheang, "Online judge," *Computers & Education*, Vol. 36, 2001, pp. 299-315.
8. H. Sun, B. Li, and M. Jiao, "YOJ: An online judge system designed for programming courses," in *Proceedings of the 9th IEEE International Conference on Computer Science and Education*, 2014, pp. 812-816.
9. S. Tang, L. Zou, and X. Liao, "A research on online judge technology based on MOOC platform," *DEStech Transactions on Engineering and Technology Research*, 2016, No. iect2016/3720.
10. G. P. Wang, S. Y. Chen, X. Yang, and R.-Y. Feng, "OJPOT: Online judge & practice oriented teaching idea in programming courses," *European Journal of Engineering Education*, Vol. 41, 2016, pp. 304-319.
11. H. Wu, Y. Liu, L. Qiu, and Y. Liu, "Online judge system and its applications in c language teaching," in *Proceedings of IEEE International Symposium on Educational Technology*, 2016, pp. 57-60.
12. J. Wu, S. Chen, and R. Yang, "Development and application of online judge system," in *Proceedings of IEEE International Symposium on Information Technologies in Medicine and Education*, Vol. 1, 2012, pp. 83-86.
13. N. Wirth, *Algorithms and Data Structures*, Pearson Education, PA, 1986.
14. R. A. Baeza-Yates, "Teaching algorithms," *ACM SIGACT News*, Vol. 26, 1995, pp. 51-59.
15. R. Lawrence, "Teaching data structures using competitive games," *IEEE Transactions on Education*, Vol. 47, 2004, pp. 459-466.
16. Y.-S. Gu and J.-Y. Zhu, "Teaching research on data structure based on knowledge structure," in *Proceedings of IEEE International Conference on Computer Science and Software Engineering*, Vol. 5, 2008, pp. 404-406.
17. S. Wasik, M. Antczak, J. Badura, A. Laskowski, and T. Sternal, "A survey on online ju-

dge systems and their applications," *ACM Computing Surveys*, Vol. 51, 2018, pp. 1-34.
18. C. A. Higgins, G. Gray, P. Symeonidis, and C. Tsintsifas, "Automated assessment and experiences of teaching programming," *Journal on Educational Resources in Computing*, Vol. 5, 2005, pp. 5-es.
19. C. M. Intisar and Y. Watanobe, "Classification of online judge programmers based on rule extraction from self organizing feature map," in *Proceedings of the 9th IEEE International Conference on Awareness Science and Technology*, 2018, pp. 313-318.
20. W. Zhou, Y. Pan, Y. Zhou, and G. Sun, "The framework of a new online judge system for programming education," in *Proceedings of ACM Turing Celebration Conference*, 2018, pp. 9-14.
21. C. K. Poon, T.-L. Wong, C. M. Tang, J. K. L. Li, Y. T. Yu, and V. C. S. Lee, "Automatic assessment via intelligent analysis of students' program output patterns," in *Proceedings of International Conference on Blended Learning*, 2018, pp. 238-250.
22. P. Denny, A. Luxton-Reilly, and D. Carpenter, "Enhancing syntax error messages appears ineffectual," in *Proceedings of Conference on Innovation and Technology in Computer Science Education*, 2014, pp. 273-278.
23. P. Denny, A. Luxton-Reilly, E. Tempero, and J. Hendrickx, "Understanding the syntax barrier for novices," in *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education*, 2011, pp. 208-212.
24. R. S. Pettit, J. Homer, and R. Gee, "Do enhanced compiler error messages help students? Results inconclusive," in *Proceedings of ACM SIGCSE Technical Symposium on Computer Science Education*, 2017, pp. 465-470.
25. M. T. Pham and T. B. Nguyen, "The DOMJudge based online judge system with plagiarism detection," in *Proceedings of IEEE-RIVF International Conference on Computing and Communication Technologies*, 2019, pp. 1-6.
26. R. Kasahara, K. Sakamoto, H. Washizaki, and Y. Fukazawa, "Applying gamification to motivate students to write high-quality code in programming assignments," in *Proceedings of ACM Conference on Innovation and Technology in Computer Science Education*, 2019, pp. 92-98.
27. T. Yamamoto, M. Matsushita, T. Kamiya, and K. Inoue, "Measuring similarity of large software systems based on source code correspondence," in *Proceedings of International Conference on Product Focused Software Process Improvement*, 2005, pp. 530-544.
28. S.-Y. Noh, "An XML plagiarism detection model for procedural programming languages," Technical Report No. TR03-14, Computer Science Department, Iowa State University Digital Repository, 2003.
29. L. Jiang, G. Misherghi, Z. Su, and S. Glondu, "Deckard: Scalable and accurate tree-based detection of code clones," in *Proceedings of the 29th IEEE International Conference on Software Engineering*, 2007, pp. 96-105.
30. E. Bendersky, "pycparser: C parser and AST generator written in python," https://github.com/eliben/pycparser, 2011.
31. I. D. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier, "Clone detection using abstract syntax trees," in *Proceedings of IEEE International Conference on Software Maintenance*, 1998, pp. 368-377.
32. A. Savvides, C.-C. Han, and M. B. Strivastava, "Dynamic fine-grained localization in ad-hoc networks of sensors," in *Proceedings of the 7th Annual International Confer-*

ence on Mobile Computing and Networking, 2001, pp. 166-179.
33. J.-P. Sheu, P.-C. Chen, and C.-S. Hsu, "A distributed localization scheme for wireless sensor networks with improved grid-scan and vector-based refinement," *IEEE Transactions on Mobile Computing*, Vol. 7, 2008, pp. 1110-1123.
34. M. L. Sichitiu and V. Ramadurai, "Localization of wireless sensor networks with a mobile beacon," in *Proceedings of IEEE International Conference on Mobile Ad-hoc and Sensor Systems*, 2004, pp. 174-183.
35. K.-F. Ssu, C.-H. Ou, and H. C. Jiau, "Localization with mobile anchor points in wireless sensor networks," *IEEE Transactions on Vehicular Technology*, Vol. 54, 2005, pp. 1187-1197.
36. M. Sugano, T. Kawazoe, Y. Ohta, and M. Murata, "Indoor localization system using RSSI measurement of wireless sensor network based on zigbee standard," *Wireless and Optical Communications*, Vol. 538, 2006, pp. 1-6.
37. G. Wang and K. Yang, "A new approach to sensor node localization using RSS measurements in wireless sensor networks," *IEEE Transactions on Wireless Communications*, Vol. 10, 2011, pp. 1389-1395.
38. Y.-H. Wu and W.-M. Chen, "Localization using a mobile beacon with directional antenna for wireless sensor networks," *IEICE Transactions on Information and Systems*, Vol. 94, 2011, pp. 2370-2377.
39. B. Xiao, H. Chen, and S. Zhou, "Distributed localization using a moving beacon in wireless sensor networks," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 19, 2008, pp. 587-600.
40. M. Boushaba, A. Hafid, and A. Benslimane, "High accuracy localization method using AoA in sensor networks," *Computer Networks*, Vol. 53, 2009, pp. 3076-3088.

Chorng-Shiuh Koong (孔崇旭) received M.S. and Ph.D. degree in Computer Science and Information Engineering from National Chiao-Tung University, Taiwan in 1995 and 2000 respectively. Currently, he is a Professor in the Department of Computer and Information Science at National Taichung University of Education, Taiwan. His research interests include software testing, object-oriented computing, software component technology, visual programmming, and E-learning technologies.



Zih-Yu Wei (魏子裕) received his M.S. from the National Taichung University of Education in 2020. His research interests include software testing, game-based learning, computer assisted instruction, and website technologies.

Chun-Hsien Chen (陳俊憲) received his B.S. from the National Chung Hsing in 1991. His research interests include game-based learning, computer assisted instruction, IoT application, and big-data in healthcare.

Cheng-Yan Guo (郭承諺) received his M.S. from National Taiwan University College of Medicine in 2017, and B.S. from the National Taichung University of Education in 2014. His research interests include software testing, robotics, computer vision, bio-signal processing, and embedded systems.