An Implementation of Java Programming Learning Assistant System in University Course

Xiqin Lu, Nobuo Funabiki, Soe Thandar Aung, Yanhui Jing
Department of Information and Communication Systems
Okayama University, Okayama, Japan
pch55zhl@s.okayama-u.ac.jp, funabiki@okayama-u.ac.jp
soethandar@s.okayama-u.ac.jp, pokn4s4h@s.okayama-u.ac.jp

Shingo Yamaguchi
Department of Information Science and Engineering
Yamaguchi University, Ube, Japan
shingo@yamaguchi-u.ac.jp

2023 11th International Conference on Information and Education Technology (ICIET) | 978-1-6654-6548-9/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICIET56899.2023.10111422

***Abstract*—Nowadays, *Java programming* is used in a variety of application systems as a highly portable object-oriented programming language. To assist its self-studies by novice students, we have developed the *Java programming learning assistant system (JPLAS)*, and implemented the personal answer platform on *Node.js*. *JPLAS* offers several types of exercise problems at different learning levels, including the *grammar-concept understanding problem (GUP)*, the *value trace problem (VTP)*, the *mistake correction problem (MCP)*, the *element fill-in-blank problem (EFP)*, the *code completion problem (CCP)*, and the *phase fill-in-blank problem (PFP)*. Any answer is automatically marked by *string matching* with the correct one on the platform. In this paper, we present an implementation of the six problem types in a Java programming course in Okayama University. We generated 109 problem instances by following its curriculum and assigned them to 58 third-year students as homework before the final examination. Their solution results reveal the difficulty difference among the problem types and confirm the validity in the Java programming course.**

***Index Terms*—Java programming, JPLAS, exercise problem, self-study, university course**

I. Introduction

Java is a powerful, general-purpose object-oriented programming language. *Java* was created in 1995, and is running on more than three billion devices worldwide as one of the most popular programming languages. Nowadays, *Java* is used in a variety of application systems, such as web applications, mobile applications, desktop applications, games, IoT application systems, and cloud service systems, as a highly portable object-oriented programming language. Thus, many universities and professional schools are offering the courses to cultivate Java programming engineers.

To assist self-studies of *Java programming* by novice students, we have developed the *Java programming learning assistant system (JPLAS)* and implemented the personal answer platform on *Node.js* that will be distributed to students on *Docker* [1].

JPLAS offers several types of exercise problems at different learning levels for different goals, to cover step-by-step self-study of *Java programming* by novice students. They include the *grammar-concept understanding problem (GUP)* [2], the *value trace problem (VTP)* [3], the *mistake correction problem (MCP)* [4], the *element fill-in-blank problem (EFP)* [5], the *code completion problem (CCP)* [6], and the *phase fill-in-blank problem (PFP)* [7].

In these problem types, one problem instance consists of a source code, a set of questions, and their correct answers. The correctness of any answer from a student is automatically marked through *string matching* with the stored correct answer. The common answer interface for them has been implemented on a web browser, where the marking function was implemented by *JavaScript* that runs on the web browser [8].

The description and the learning goal of each problem are listed:

In a *GUP* instance, each question describes a basic grammar concept on a reserved word or a common library function as *keyword* in *Java programming*, and requests to answer the corresponding element appearing in the source code. *GUP* aims the basic *grammar* study.

In a *VTP* instance, the given source code contains several standard output statements for important variables or output messages, and each question requests to answer the output value. *VTP* aims the *code reading* study [9].

In an *MCP* instance, the source code contains several mistaken elements of reserved words or common library functions in *Java programming*, and requests to answer any mistaken element and its correction in the source code. *MCP* aims the *code debugging* study.

In an *EFP* instance, the source code contains several blank elements while specifying their locations. A blank element can be a reserved word, an identifier, an operator, and a control symbol. It requests to fill in every blank by the original element in the source code. *EFP* aims the first-step *code writing* study.

In a *CCP* instance, the source code contains several blank elements like *EFP*, but does not specify their locations. Then, it requests to find the location of every missing element in the source code and fill in it with the original one. *CCP* aims the *code reading & writing* study.

In a *PFP* instance, the source code contains several blanks of phrases, or sets of multiple elements, and requested to fill in each blank by the original set of elements or the message in the source code. Unlike *EFP* where one blank can substitute only one element, one blank in *PFP* can substitute any number elements or text messages in the source code. This flexibility can increase the candidates for blanks in the source code and enhance the difficulty level of problem solutions by novice students. *PFP* aims the *code writing* study.

Unfortunately, *JPLAS* has not been implemented in a Java programming course in a university with the designated exercise problems for the course. Thus, the validity and effectiveness of *JPLAS* in the course application need to be verified.

In this paper, we present an implementation of the six problem types in *JPLAS* in a Java programming course in Okayama University. In this implementation, we generated a total of 109 problem instances by following its curriculum. Specifically, they include 28 for *GUP*, 15 for *VTP*, 12 for *MCP*, 26 for *EFP*, 13 for *CCP*, and 15 for *PFP*. We assigned them to 58 third-year students in the course as homework, and asked them to submit the answer text files at the e-learning system *Moodle* before the final examination. The answer text file contain every answer submission record including the date and time, the instance number, and the student answer and its marking result for each question. In each class, at least three teaching assistants helped the students for installing the personal answer platform and solving the problems.

After the final examination, the answer results of the students are analyzed by running the *answer analyzer* with the submitted answer text files from the students. The *answer analyzer* was implemented by *Java*. It selects the final answers of the students to each instance, calculates the average correct answer rate and the number of submission times for each problem and for each student, and outputs the results into an Excel file. Their answer results reveal that the solution performance is low at *CCP* and *PFP*. We also analyze the correlation between the average correct answer rate of *JPLAS* and the final examination. The correlation coefficient between them is 0.75. It means if students can understand exercise problems in *JPLAS* well, they can have good grades in the final examination. Thus, the validity of the application in the Java programming course was confirmed.

The rest of this paper is organized as follows: Section II discusses related works in literature. Section III reviews our preliminary works on *JPLAS*. Section IV presents the implementation of the generated exercise problems. Section V shows the implementation results to novice students and discussions. Finally, Section VI concludes this paper with future work.

II. Related Works

In this section, we discuss related works in literature.

In [10], Okimoto et al. developed an educational support system that can automatically generate a source code to facilitate the programming instruction through *code reading*. The system requires a learner to answer the value of a variable after the execution of the code. They applied this system to 108 first-year students in a programming course, and found that the *code reading* comprehension is difficult for programming beginners.

In [11], Suzuki et al. proposed a web-based learning support system for classroom teaching called *ClassCode*. It provides an environment where students can follow tutorials of interactive coding exercises that are intertwined with their learning paces, while teachers can outline how they are learning.

III. Review of JPLAS

In this section, we review our preliminary works on *JPLAS*.

A. Personal Answer Platform

JPLAS is a web application system that allows a teacher to offer programming exercises to students, and allows a student to solve them by himself/herself with the automatic marking function. *Node.js* [12] is used for the web application platform, and *JavaScript* and *Java* are used in the application programs, and the data is stored in the file system. *Docker* [13] is used to distribute the system to students. We prepared the manual to explain the installation of the system into a PC and the usage instructions of solving problems and submitting answer text files.

B. Exercise Problems in JPLAS

JPLAS offers several exercise problem types to cover self-study of *Java programming* at different levels by novice students. In this paper, the six types of *GUP*, *VTP*, *MCP*, *EFP*, *CCP*, and *PFP* are considered in the course implementation. Basically, the difficulty level increases by this order.

1) Grammar-concept Understanding Problem: *GUP* reminds the knowledge and concepts of reserved words and common library functions. Each question describes the concept of one keyword appearing in the source code. *GUP* is the easiest problem in *JPLAS*, since a student only needs to find the corresponding keyword in the source code to each question. As an example *GUP* instance, TXT1 shows the source code and the six questions with six answer forms.

TXT1: Example GUP instance.

```
1 public class FirstProgram {
2   public static void main(String[] args) {

3     System.out.print ("HelloJava");
4   }
5 }

Question:
Q1. What is access modifier in Line 1?
     １行目のアクセス修飾子とは何ですか？  _1_
Q2. What is class name?
    クラス名とは何ですか？  _2_
Q3. Which keyword allows the method to run
    without creating an object?
    オブジェクトを作成せずにメソッドを実行できるようにす
    るキーワードは何ですか？  _3_
Q4. Which keyword describes no returning data
    in Line 2?
    ２行目でデータを返さないことを記述しているキーワード
    は何ですか？  _4_
Q5. Which data type is used in Line 2?
    ２行目で使用されているデータ型は何ですか？  _5_
Q6. What will be displayed as the output?
    出力として何が表示されるのでしょうか？  _6_
```

2) Value Trace Problem: VTP questions the values of important variables and output messages in the source code. To prevent a student from running the source code to know the outputs by copying it into a text file, the code in the HTML file cannot be copied in *VTP*. As an example *VTP* instance, TXT2 depicts the source code and the three questions with seven answer forms. This instance is made for studying the `for` loop and `if` statement.

TXT2: Example VTP instance.

```
import java.io.*;
class Sample{
   public static void main(String[] args){
      boolean bl = false;
      for(int i=0; i<5; i++){
         for(int j=0; j<5; j++){
            if(bl == false){
               System.out.print("*");
               bl = true;
            }
            else{
               System.out.print("-");
               bl = false;
            }
         }
         System.out.print("\n");
      }
   }
}

Question:
Q1:If bl is true, what will be outputted?
    _1_
Q2:if bl is false, what will be outputted?
    _2_
Q3:What is the standard output?
  _3_
  _4_
  _5_
  _6_
  _7_
```

3) Mistake Correction Problem: MCP requests to answer every mistaken element and its correction in the given corrupt source code. The hint function is implemented here to avoid easily giving up solving the instance. When a student clicks the hint button, the first character of the correct answer for each question will appear. Besides, to prevent the easy use of the hint function, the button can be clicked only when the answering time exceeds five minutes and the number of answer submission times exceeds five. As an example of *MCP* instance, TXT3 depicts the source code and the questions with six answer forms.

TXT3: Example MCP instance.

```
1:import java.io.;
2:class Sample{
3:   public static void main(String[] args)
    throws IOException{
4:       System.out.println("何番目のコースにします
    か？");
5:       System.out.println("整数を入力してください。
    ");
6:     BufferedReader br = native
    BufferedReader(new InputStreamReader(
    System.in));
7:       String str = br.readLine();
8:       int res = Character.parseInt(str);
9:       char ans = (res = 1) ? 'A' : 'B';
10:    System.out.println(ans+"コースを選択しまし
    た。");
11:  }
12:}

Question:
line#: incorrect -> correct
7:  _1_ -> _2_
8:  _3_ -> _4_
9:  _5_ -> _6_
```

4) Element Fill-in-blank Problem: The source code in an *EFP* instance is not a complete one, unlike the code for the three types of problems introduced before. It requests to answer the blank elements in the source code by understanding syntax and semantics. As an example *EFP* instance, TXT 4 depicts the source code and the questions with 13 answer forms. It was made for studying the `for` loop.

TXT4: Example EFP instance.

```
_1_ java.io.*;
_2_ Sample{
   public _3_ void _4_(String[] args) _5_
       IOException{
      System.out.println("いくつ*を出力しますか?");
      BufferedReader _6_  =
        _7_ BufferedReader(_8_
            InputStreamReader(System.in));
      String _9_ = br.readLine();
      _10_ num = Integer.parseInt_11_ str);
      _12_(int i = 1; _13_ <= num; i++){
         System.out.print("*");
      }
   }
}
```

5) Code Completion Problem: CCP requests to answer the blank elements in the given source code like *EFP*. However, *CCP* does not show the blank locations in the code. As an example *CCP* instance, TXT 5 depicts the uncompleted source code. It was made for studying the `switch` loop. A students needs to add the missing element at the proper location in the code.

TXT5: Example CCP instance.

```
1: java.io.*;
2:class Sample{
3:   public static void main(String[] ) throws
     IOException{
4:      .out.println("Please enter a or b.");
5:      BufferedReader br =  new
    BufferedReader(new (System.in));
6:      String str = .readLine();
7:       res = str.charAt(0);
8:      switch(res){
        'a':
           System.out.println("a was entered.");
           break;
        case 'b':
           System.out.println("b was entered.");
           ;
        :
           System..println("Please enter a or b.");
           break;
     }
  }
}
```

6) *Phase Fill-in-blank Problem: PFP* requests to answer the blank elements in the source code like *EFP*. However, differing from *EFP* where one blank consists of only one element, one blank in *PFP* can substitute any number elements or text messages in the source code. To guide the correct answer, the necessary descriptions such as the inputs and outputs are given together for each PFP instance. Since the correct answer for one blank may not be unique, the answer platform for *PFP* can handle up to two correct answers. As an example *PFP* instance, TXT 6 depicts the source code and the questions with 10 answer forms. It was made for studying *input data from keyboard*.

TXT6: Example PFP instance.

```
import  _1_ ;
class Sample{
   public  _2_ {
      System.out.println( _3_ );
      BufferedReader br = new  _4_ ;
      String str1 =  _5_ ;
      String str2 =  _6_ ;
      int num1 =  _7_ ;
      int num2 =  _8_ ;
      System.out.println( _9_ );
      System.out.println( _10_ );
   }
}
```

C. Answer Analyzer

After a student completes answering the given exercise problems, he/she will submit the answer text file to the teacher. This file contains the submission date and time, the instance ID, and the answer and its marking result of each question at every answer submission by this student. After collecting the answer files from the students by emails or a file server, the teacher can calculate the average correct answer rate and the average number of submission times by running the *answer analyzer*.

IV. Generated Exercise Problems

In this section, we present the generated exercise problems in *JPLAS* for the course implementation.

A. Generated Exercise Problems

A total of 109 problem instances for the six types were generated for *JPLAS* by following the curriculum of the Java programming course. The original sources codes for the instances were selected from the sample source codes provided with the textbook used in the class [14]. Table I shows the topics of 16 chapters in the textbook, the corresponding instance number of each problem type for each chapter, and the number of instances for each problem type with the total.

TABLE I: Generated problem instances.

topic in textbook	GUP	VTP	MCP	EFP	CCP	PFP
	corresponding instance ID					
1: standard output	1	1	-	1	1	1
2: basic of Java	2	2	1	2	-	2
3: variable	3-6	3	2	3,4	2	3
4: expression and operator	7-9	4	3	5,6	3	4
5: if-else and switch	10,11	5	4	10	4	5
6: do-while and for	12,13	6	5	8,9	5	6
7: array	14	7	-	11,12	6	7
8: basic of class and method	15,16	8	6	13,14	7	8
9: function of class and structure	17	9	7	15,16	8	9
10: use of class	18	10	8	17,18	9	10
11: inheritance	19	11	-	19,20	10	11
12: interface	20	12	9	21	11	12
13: import and package	21-24	13	10	22	-	-
14: exceptions and I/O processing	25-28	14	-	23	12	13
15: thread manipulation	-	15	11	24	13	14
16: graphical application	-	-	12	25,26	-	15
# of instances	28	15	12	26	13	15
total # of instances	109					

B. Generation of Instance

An instance in *JPLAS* can be generated by the following procedure:

1) To select a proper source code,
2) To generate the text file as shown in TXT 1 - TXT 6 by running the corresponding *generators* that we have implemented.
3) To generate the CSS/HTML/JavaScript files for the answer interface on the web browser by running the *generator* with this text file that we have also implemented,
4) To install the generated instances to the answer platform.

C. Use of JPLAS

A student can solve the *JPLAS* instances by the following procedure:

1) Download and install *Docker* in the PC.
2) Download the *JPLAS Docker image* from *Docker hub*.
3) Run the *JPLAS Docker image* in the PC by using the *Docker run* command.
4) Open the browser and type *localhost:4000* to access to the *JPLAS* platform.

V. Evaluation

In this section, we present the implementation results of the 109 instances to 58 third-year students taking a Java programming course in Okayama University.

A. Solution Performances by Problem Types

First, we analyze differences of solution performances of the students by the six problem types. Table II shows the number of students who submitted answer files, the average correct answer rate, and the average number of answer submission times for each instance by them for each problem type. From this table, it can be observed that *GUP*, *VTP*, *MCP*, and *EFP* are easier for these novice students. The average correct answer rate and the average number of answer submission times are similar in them. *CCP* and *PFP* are harder for them, where both the correct rate and the number of submission times become worse. The average correct answer rate of two *CCP* instances at ID = 8 and 10, and one *PFP* instance at ID = 12 is less than 90%. According to Table I, they are on `override`, `overload`, `structure`, `inheritance`, `abstract class`, and `interface`, which can be difficult for novices to understand. The hint function should be implemented for them, which will be in future works.

TABLE II: Summary of solution results by problem types.

problem type	# of instances	# of submitted students	average correct rate (%)	average #of submissions
GUP	28	46	99.83	3.47
VTP	15	45	99.70	2.25
MCP	12	46	97.93	3.01
EFP	26	45	99.76	2.93
CCP	13	45	90.46	6.32
PFP	15	45	94.61	6.55
average			97.04	4.09

B. Solution Performances by Students

Next, we analyze differences of solution performances by 44 students who submitted answer files to all of the six problem types to confirm the validity of the application in the Java programming course. Figure 1 shows the average number of answer submission times and the average correct answer rate for the *GUP*, *VTP*, *MCP*, *EFP*, *CCP*, and *PFP* instances by the individual students. Besides, Figure 2 shows the average number of answer submission times and the average correct answer rate among all the instances by the individual students.

From these results, 21 students among 44 at ID = 3, 4, 6, 8, 11, 12, 13, 15, 17, 19, 20, 21, 22, 23, 24, 26, 30, 31, 33, 35, and 37, achieved the 100% correct rate in any instance. These students well studied all the topics in the Java programming course. Besides, among the 21 students, three students at ID = 20, 30, and 33 also have low number of submission times, which is less than two at any instance. They are excellent students.

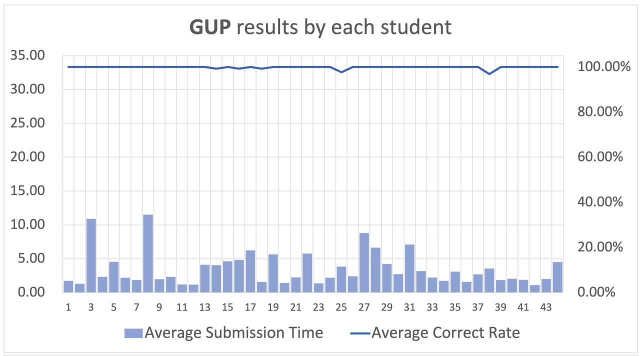
On the other hand, the average number of submission times by three students at ID = 8, 19, and 31 are more than seven, which means that they tried hard to reach the correct answers. They are diligent students. 22 students among 44 achieved the higher than 90% average correct rate. Only one student did not reach 90%. These results suggest that the generated instances in the six problem types are proper for novice students to solve Java programming exercise problems as homework.

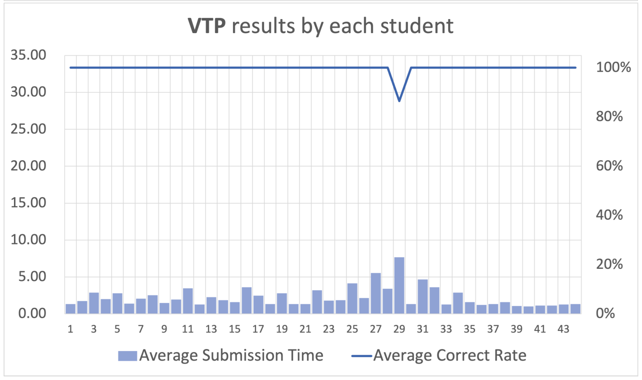
C. The correlation between JPLAS and final examination results

To confirm the learning effectiveness and the suitability of *JPLAS* for novice students, we calculate the correlation coefficient of JPLAS and final examination results by students.

The final examination contains two *code writing problem(CWP)* assignments, which is one type of exercise problem in JPLAS. One assignment in *CWP* consists of the statement and the *test code* that should be prepared by a teacher. A student is requested to write the Java source code that passes every test case described in the test code. The correctness of the source code written by the student is verified by running the test code with the source code on *JUnit* for *code testing* [15]. Thus, the *CWP* is more difficult than the six type of exercise problems that introduced before.

The correlation coefficient is a measure of the strength of the linear relationship between two sets of data. It takes values in the range −1˜+1. If it is in the range of 0.7˜1.0, there is a strong correlation. if it is in the range of 0˜0.2, there is almost no correlation. The correlation coefficient between the average correct answer rate of *JPLAS* and the final examination is 0.75. From the result, we can observe that students who understand exercise problems in *JPLAS* well can have good grades in the final examination. Thus, the effectiveness of our proposal is confirmed.





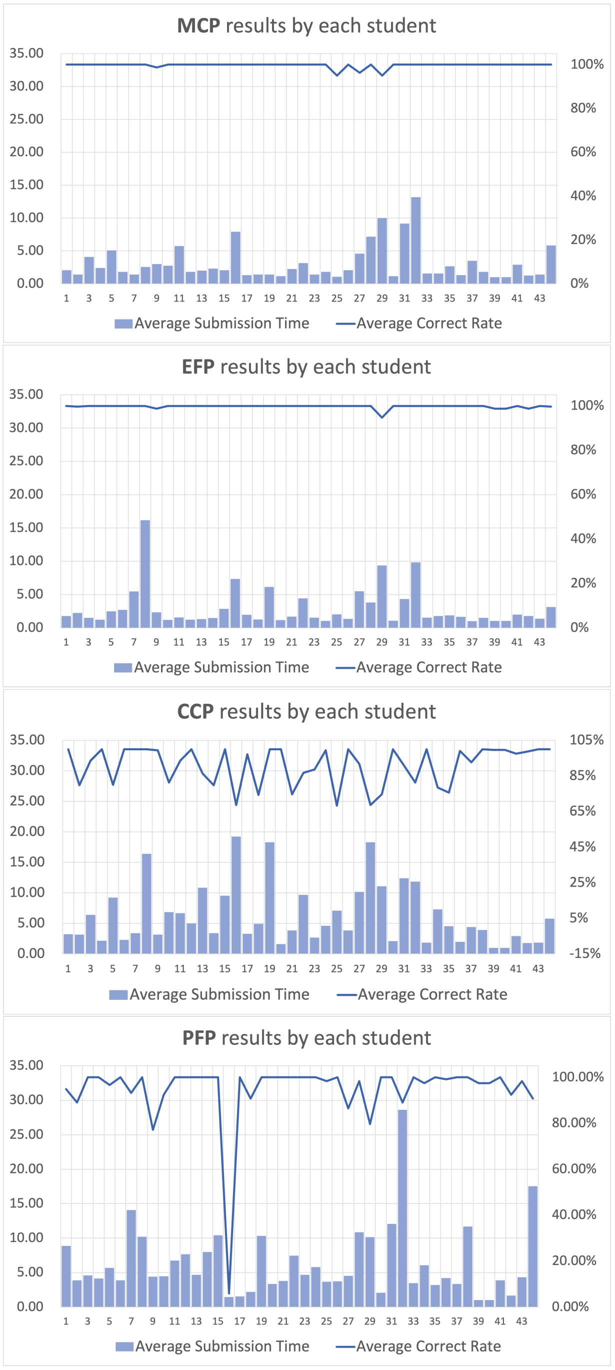


Fig. 1: Results of each student for each problem type.

VI. Conclusion

This paper presented the implementation of *GUP*, *VTP*, *MCP*, *EFP*, *CCP*, and *PFP* instances in *Java programming learning assistant system (JPLAS)* to a Java programming course in Okayama University. A total of 109 instances were generated by following the course curriculum, and were assigned to 58 third-year students as homework. The solution results revealed the difficulty difference among the six problem types and confirmed the validity in the course. In future works, we will implement the hint function for hard problem types, generate new instances on other topics, and apply them to students in various universities and departments.

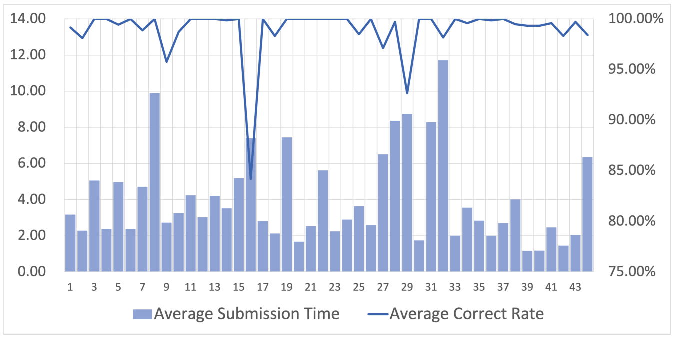


Fig. 2: Average solution results of each student among all problem types.

References

[1] S. T. Aung, N. Funabiki, L. H. Aung, H. Htet, H. H. S. Kyaw, and S. Sugawara, "An implementation of Java programming learning assistant system platform using Node.js," in Proc. ICIET, pp. 47-52, April.2022.

[2] X. Lu, N. Funabiki, S. T. Aung, H. H. S. Kyaw, K. Ueda, and W-C. Kao, "A study of grammar-concept understanding problem in C programming learning assistant system," ITE Trans. Media Tech. Appl., vol. 10, no. 4, pp. 198-207, Oct. 2022.

[3] X. Lu, N. Funabiki, H. H. S. Kyaw, E. E. Htet, S. L. Aung, and N. K. Dim, "Value trace problems for code reading study in C programming," Adv. Sci. Tech. Eng. Syst. J. (ASTESJ), vol. 7, no. 1, pp. 14-26, Jan. 2022.

[4] Y. Jing, N. Funabiki, S. T. Aung, X. Lu, A. A. Puspitasari, H. H. S. Kyaw, and W-C. Kao, "A proposal of mistake correction problem for debugging study in C programming learning assistant system," Int. J. Info. Edu. Tech. (IJIET), vol. 12, no. 11, pp. 1158-1163, Oct.2022.

[5] N. Funabiki, Tana, K. K. Zaw, N. Ishihara, and W.-C. Kao, "A graph-based blank element selection algorithm for fill-in-blank problems in Java programming learning assistant system," IAENG Int. J. Comp. Sci., vol. 44, no. 2, pp. 247-260, May 2017.

[6] H. H. S. Kyaw, S. S. Wint, N. Funabiki, and W.-C. Kao, "A code completion problem in Java programming learning assistant system," IAENG Int. J. Comp. Sci., vol. 47, no. 3, pp. 350-359, Aug. 2020.

[7] X. Lu, S. Chen, N. Funabiki, M. Kuribayashi, and K. Ueda, "A proposal of phrase fill-in-blank problem for learning recursive function in C programming," in Proc. LifeTech, pp. 127-128, March 2022.

[8] N. Funabiki, H. Masaoka, N. Ishihara, I-W. Lai, and W.-C. Kao, "Offline answering function for fill-in-blank problems in Java programming learning assistant system," in Proc. ICCE-TW, pp. 324-325, May 2016.

[9] Coder ' s Cat, "Learn from source code (an effective way to grow for beginners)," https://medium.com/better-programming/learn-from-source-code-an-effective-way-to-grow-for-beginners-e0979e9b5a84, Nov. 2019.

[10] K. Okimoto, S. Matsumoto, S. Yamagishi, and T. Kashima, "Developing a source code reading tutorial system and analyzing its learning log data with multiple classification analysis," Art. Life Robot. vol. 22, pp. 227-237, 2017.

[11] R. Suzuki, J. Kato, and K. Yatani, "ClassCode: an interactive teaching and learning environment for programming education in classrooms," arXiv:2001.08194 [cs.CY], Jan. 2020.

[12] D. Herron, Node.js web development, Packt Pub., 2016.

[13] R. McKendrick, Monitoring Docker, Packt Pub., 2015.

[14] M. Takahashi, Yasashii Java, ver. 7, SB Creative, 2019.

[15] K. H. Wai, N. Funabiki, S. T. Aung, K. T. Mon, H. H. S. Kyaw, and W.-C. Kao, "An Implementation of Answer Code Validation Program for Code Writing Problem in Java Programming Learning Assistant System," in ICIET, March 2023.