A Proposal of Hint Function for Java Programming Learning Assistant System

Yanhui Jing, Nobuo Funabiki*, Soe Thandar Aung, Xiqin Lu, Htoo Htoo Sandi Kyaw, and Kiyoshi Ueda

***Abstract*—Nowadays, Java is one of the world's most used object-oriented programming languages for its dependability and portability. To assist the self-studies of novice students, we have developed the Java programming learning assistance system (JPLAS). JPLAS offers several types of exercise problems for different difficulties, including the *grammar-concept understanding problem (GUP)*, the *mistake correction problem (MCP)*, the *element fill-in-blank problem (EFP)*, and *the phrase fill-in-blank problem (PFP)*, where a question asks to answer the corresponding keyword or phrase in the given source code. Unfortunately, some students cannot solve them well as the difficulty level is ascending. In this paper, to help such students, we propose a hint function that will show the first or more characters of each correct answer word in the answer interface on a web browser for JPLAS, when requested. The use of this function by a student can be traced by a teacher to analyze the performance and problem difficulty. For evaluations, we generated three instances for GUP, MCP, EFP, and PFP respectively, and assigned them to junior students taking the Java programming course in Okayama University, Japan. Their answer results found that the proposal is effective in helping students solve exercise problems in JPLAS, but the thinking time needs to be adjusted for PFP.**

***Index Terms*—C programming, CPLAS, mistake correction problem, automatic generation, answer interface**

I. Introduction

For decades, *Java* has been widely used in industries as the dependable and portable object-oriented programming language. *Java* has also been implemented in mission-critical systems for major organizations as well as in small-sized embedded systems. Thus, there has been a high demand among IT firms on educations of Java programming designers. Actually, a great number of universities and professional schools have offered *Java programming* courses.

To enhance Java programming educations, we have developed a web-based *Java programming learning assistant system (JPLAS)* [1, 2], JPLAS provides a variety of exercise problems that may be used to gradually progress the learning stages and cover self-studies of Java programming at various levels by novice students. In any instance of these problems, an answer from a student is automatically marked in the system. Therefore, a student can continue solving the given problems until all the answers become correct.

Actually, JPLAS offers the grammar-concept understanding problem (GUP) [3], the mistake correction problem (MCP) [4], the value trace problem (VTP) [5], the element fill-in blank problem (EFP) [6], the code completion problem (CCP) [7], the phrase fill-in-blank problem (PFP) [8], and the code writing problem (CWP) [9]. In any instance of these problems, the correctness of an answer from a student is automatically checked in JPLAS. For GUP, MCP, VTP, EFP, and CCP, it is checked through string matching with the correct one stored in the system. For CWP, it is verified through unit testing by running the test code on JUnit. Among these problems, this paper is targeting GUP, MCP, EFP, and PFP in JPLAS, since a new instance can be generated automatically by running our implemented instance generator, once the proper source code is selected.

The difficulty level will be increased as this order. Therefore, we have found that more students cannot solve PFP in previous implementations to them. To make them continue solving the questions, some assistance functions will be necessary when they meet difficulties.

In this paper, we propose and implement a *hint function* in the answer interface on a web browser for the four problems of GUP, MCP, EFP, and PFP in JPLAS. This function will show the *first character* of the correct answer word for the question where the answer of the student is not correct, when he/she requests it by clicking the corresponding button. Besides, it will show the correct answers to the questions in the instance, if the student cannot reach them after a lot of trials.

For PFP, our previous applications to students found that the first character of the answer word is not a sufficient hint for a student to reach the correct answer. A question in PFP asks multiple words called the *phrase* at once for each question. Therefore, we modify the hint function to make two steps. In the first step, the first character of the correct answer is shown. Besides, the other characters are also shown after being replaced by ***** to hide them, so that a student can know the number of characters in the answer. In the second step, ***** is replaced by the corresponding character in the correct answer randomly with 30%. Therefore, approximately, one-third of the answer word can be noticed by the student.

To avoid a student relying on the hint function from the beginning, the hint button can appear only after he/she clicks the answer button five times and five minutes have passed since the page is accessed.

To allow a teacher to know whether each student used the hint function at each instance, its use will be recorded and be written as the hint function flag in the answer text file that will be submitted to a teacher for grading. By checking the flag, the teacher can evaluate the performance of each student by

Manuscript received March 27, 2023; revised April 15, 2023; accepted May 8, 2023.

Yanhui Jing, Nobuo Funabiki*, Soe Thandar Aung, Xiqin Lu are with the Department of Information and Communication Systems, Okayama University, Okayama, Japan. E-mail: pf709l29@s.okayama-u.ac.jp (Y.J.)

Htoo Htoo Sandi Kyaw is with Division of Advanced Information Technology and Computer Science, Tokyo University of Agriculture and Technology, Tokyo, Japan. E-mail: htoohtoosk@go.tuat.ac.jp (H.H.S.)

Kiyoshi Ueda is with the Department of of Computer Science Nihon University, Koriyama, Japan. E-mail: ueda.kiyoshi@nihon-u.ac.jp (K.U.)

*Correspondence: funabiki@okayama-u.ac.jp (N.F.)

considering the use of the hint function. The teacher can also estimate the difficulty level of each instance by counting the number of students who used the hint function.

To implement the proposed hint function in the answer interface, we extended the *answer interface generator* so that the generated HTML/CSS/JavaScript files include the necessary button, the output field, and the hint display function. Besides, we extended the *student answer analyzer*, which will make the summary of solving results from the answer text files submitted by students, considering the use of the hint function by the individual students.

For evaluations, we generated three instances for GUP, MCP, EFP, and PFP individually, and assigned them to junior students taking the Java programming course in Okayama University, Japan. Their answer results found that the proposal is effective in helping students solving exercise problems in JPLAS, but the *solving time* needs to be adjusted for PFP.

In addition, we present a *hint function* for *code completion problem (CCP)* by extending the one for PFP. A question in CCP asks to complete the given source code that has several blank elements and incorrect ones. At the answer marking, each whole statement is compared with the corresponding correct one through *string matching*. The evaluation of the hint function for CCP will be in future studies.

The rest of this paper is organized as follows: Section II discusses related works in literature. Section III reviews our preliminary works on PLAS. Section IV presents the hint function for four problem types. Section V evaluates the hint function. Section VI presents the hint function for CCP. Finally, Section VII concludes this paper with future works.

II. Related Works in Literature

In this section, we discuss related works in literature on the hint function in programming by novice students.

Yi *et al*. [10] implemented a new repair strategy in automated program repair (APR), which is similar to the hint generation policy used in the previous intelligent tutoring system for programming (ITSP). This new policy allows partial corrections that address a portion of failed tests. However, novice students do not know how to effectively use the generated repairs as hints.

Marwan *et al*. [11] suggested that a growing body of works has explored how to automatically generate hints for novice programmers. They explored the efficacy of next-step code hints that have two complementary features: textual explanations and self-explanation prompts. They discovered that code hints with textual explanations dramatically increased their immediate programming performances.

Price *et al*. [12] presented the *QualityScore* program as a new method for automatically assessing and comparing the quality of the next programming hints using expert ratings, whereas few assessments directly evaluate or compare quality of different hint generation methods. The *QualityScore* program was used to compare the quality of the six data-driven next hint generation algorithms. The results show that although there are significant differences in quality between the six algorithms, they are relatively consistent across the different data sets and problems.

Rubinstein *et al*. [13] employed a machine learning platform called *Sense Education* in the introduction CS course. It delivers hints in real time while students work on solutions, as well as detailed feedback on submissions. It also gives a "bird's eye" picture of current capabilities and misconceptions of the class, and is biased in problem-solving approaches to train teachers how to use these technologies successfully.

Serth *et al*. [14] discussed the findings from providing contextual recommendations in a web-based development environment used for practical programming activities. They discovered that individuals requesting tips took substantially longer time and used the platform's assistance features more than the users in the control group. The findings can be used to make more particular hints to beginners, and provide additional and context-specific clues as part of the learning materials.

Fein *et al*. [15] suggested that when learning programming, to take the first step can be difficult. It is desirable for a teacher to use a system that automatically generates prompts to tell them what steps to take in the next programming task. *Catnip* is the tool that generates next step prompts for *Scratch* programming. *Catnip* uses extensive post-processing to improve the generated prompts, and displays them directly within the *Scratch* framework.

III. Review of PLAS

In this section, we review the answer platform, and the four exercise problems of GUP, MCP, EFP, and PFP as our preliminary works on JPLAS.

A. Overview of JPLAS Platform

The answer interface platform for *JPLAS* was newly implemented using *Node.js* [16] as the web application server the under the uniform design [2]. *Node.js* allows implementations of the application programs for both the server and client sides using *JavaScript*. *Express.js* [17] and *EJS* [18] are adopted together. Besides, *Docker* [19] is used for easy and correct distributions of the platform to students.

As programming languages for the implementation of the *MVC model-based* web application system, *Java* is used for the *model (M)* to run *JUnit* for testing the answer source codes from students, *EJS/CSS/JavaScript* are for *view (V)*, and *JavaScript* is for *controller (C)*, as illustrated in Fig. 1. The file system is directly used for managing data, where no database system is included.

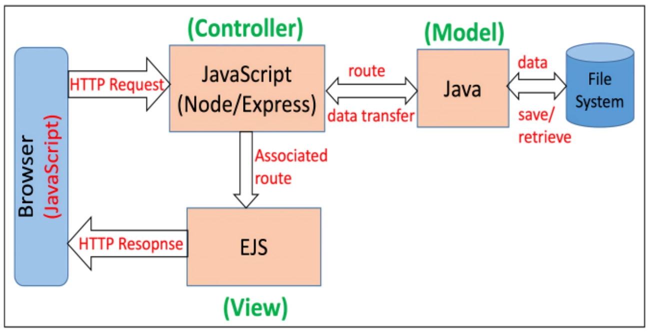
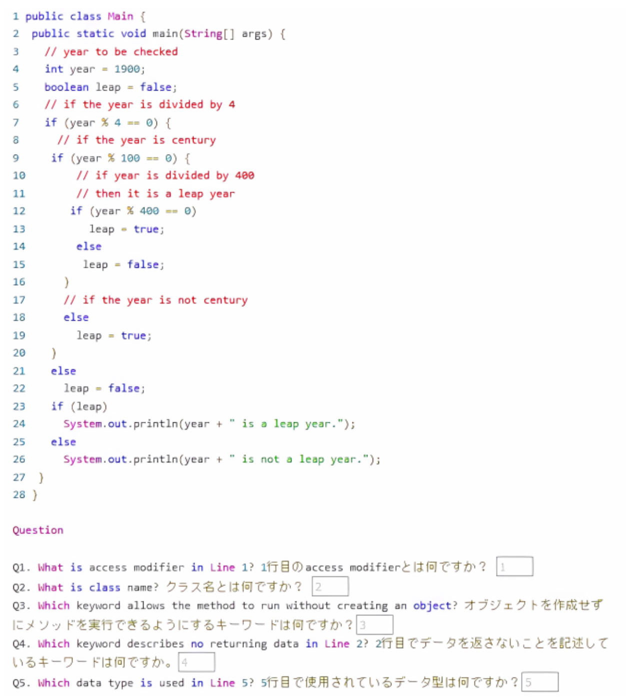


Fig. 1. Answer interface platform implementation with MVC model.

B. Grammar-Concept Understanding Problem (GUP)

A question in a GUP instance requests to answer an important word and a common library class/function in the programming language as the first-step study of basic grammar concepts. Fig. 2 illustrates the answer interface for an example GUP instance. The source code checks if the given year is a leap year or not by using a **if else** statement. A student is requested to answer the element in the source code whose definition is described in the question.



```
1 public class Main {
2  public static void main(String[] args) {
3    // year to be checked
4    int year = 1900;
5    boolean leap = false;
6    // if the year is divided by 4
7    if (year % 4 == 0) {
8      // if the year is century
9     if (year % 100 == 0) {
10        // if year is divided by 400
11        // then it is a leap year
12       if (year % 400 == 0)
13          leap = true;
14        else
15         leap = false;
16      }
17      // if the year is not century
18      else
19        leap = true;
20    }
21    else
22      leap = false;
23    if (leap)
24      System.out.println(year + " is a leap year.");
25    else
26      System.out.println(year + " is not a leap year.");
27  }
28 }
```

Question

Q1. What is access modifier in Line 1? 1行目のaccess modifier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 5? 5行目で使用されているデータ型は何ですか？[5]

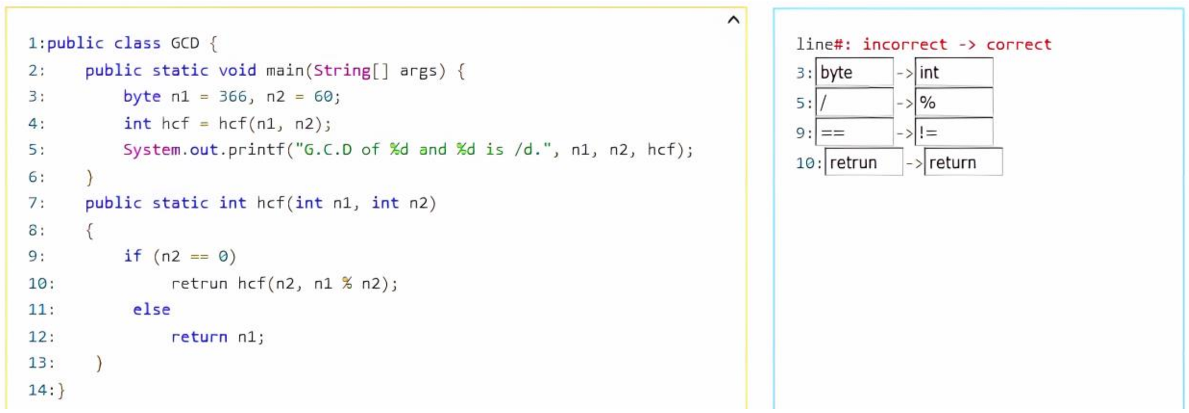
Fig. 2. Answer interface for GUP.

C. Mistake Correction Problem (MCP)

A question in a MCP instance requests to answer each mistaken element and its correction in the given corrupt source code for *code debugging* study. Fig. 3 shows the answer interface for an example MCP instance. The source code in the left column has four mistakes: **byte** at line 3 should be **int**, **/** at line 5 be **%**, **==** at line 9 be **!=**, and **retrun** at line 10 should be **return**. In the right column, the corresponding line number is shown with each pair of input forms to answer the mistaken element and the correct element respectively.

D. Element Fill-in Blank Problem (EFP)

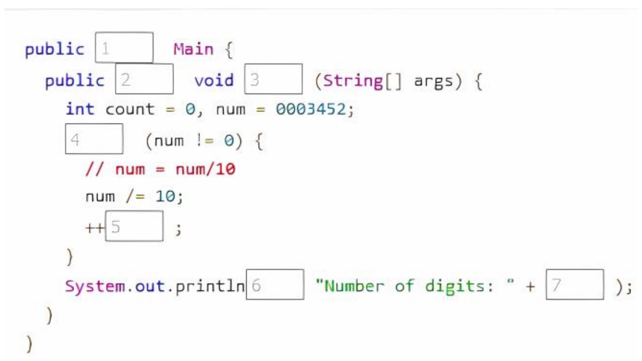
A question in a EFP instance requests to fill in the blank elements in the following source code with their originals by understanding the syntax and semantics. Fig. 4 shows the answer interface for an example EFP instance. In this source code, the **while** loop is iterated until the test expression **num != 0** is evaluated to be *false*. Following the initial iteration, the value of *num* will be 345 after being divided by 10. The count is then increased by 1. After the second iteration, the value of *num* will be 34 and the count is incremented to 2. After the third iteration, the *value* of num will be 3 and the count is incremented to 3. After the fourth iteration, the *value* of num will be 0 and the count is incremented to 4. The loop ends here because the test expression is determined to be *false*. A student is requested to fill in each blank by carefully reading the source code.



```
1:public class GCD {
2:    public static void main(String[] args) {
3:        byte n1 = 366, n2 = 60;
4:        int hcf = hcf(n1, n2);
5:        System.out.printf("G.C.D of %d and %d is /d.", n1, n2, hcf);
6:    }
7:    public static int hcf(int n1, int n2)
8:    {
9:        if (n2 == 0)
10:            retrun hcf(n2, n1 % n2);
11:         else
12:            return n1;
13:     }
14:}
```

```
line#: incorrect -> correct
3:byte    ->int
5:/       ->%
9:==      ->!=
10:retrun ->return
```

Fig. 3. Answer interface for MCP.

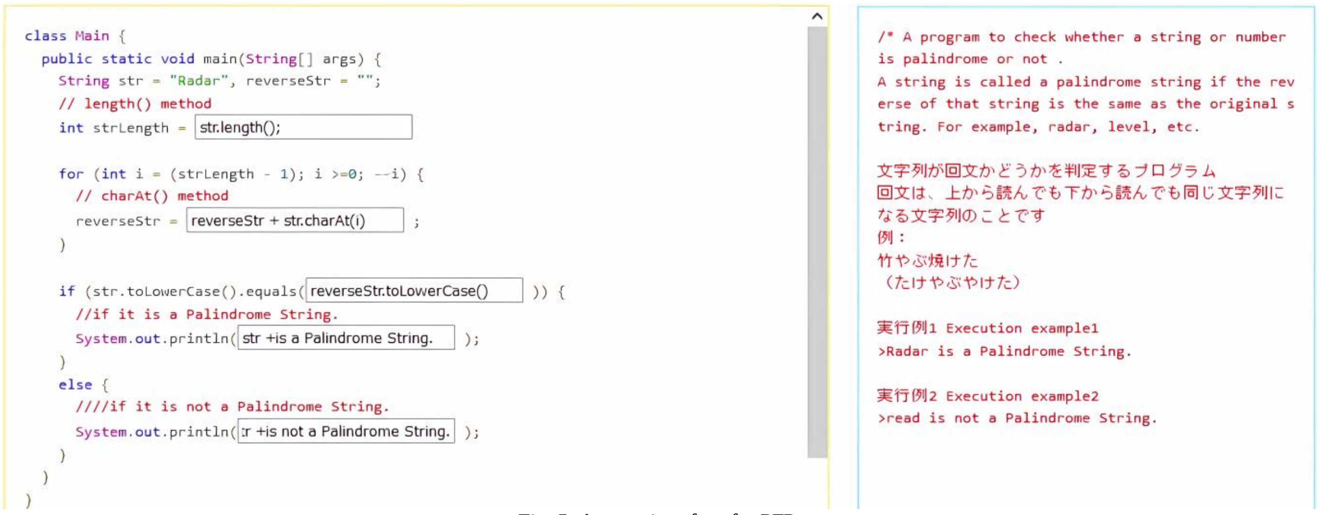


```
public [1]  Main {
  public [2]  void [3]  (String[] args) {
    int count = 0, num = 0003452;
    [4]  (num != 0) {
      // num = num/10
      num /= 10;
      ++[5] ;
    }
    System.out.println[6]  "Number of digits: " + [7]  );
  }
}
```

Fig. 4. Answer interface for EFP.

E. Phrase Fill-in-Blank Problem (PFP)

A question in a PFP instance requests to answer the corresponding phrases in the vacant part of the source code to make it complete. Fig. 5 illustrates the answer interface for an example PFP instance. The source code in the left column checks if a given string or number is *palindrome* or not. A string is considered to be a *palindrome* if both are the same when it is read from left to right or read from right to left. The source code first reverses the string or number. Then, it compares the reversed ones with the original value. The string **Radar** is stored in **str** at last. A student is requested to fill in each blank by carefully reading the source code and the statements in the right column.



```
class Main {
  public static void main(String[] args) {
    String str = "Radar", reverseStr = "";
    // length() method
    int strLength = [str.length();                ]

    for (int i = (strLength - 1); i >=0; --i) {
      // charAt() method
      reverseStr = [reverseStr + str.charAt(i)    ] ;
    }

    if (str.toLowerCase().equals([reverseStr.toLowerCase()    ] )) {
      //if it is a Palindrome String.
      System.out.println([str +is a Palindrome String.  ] );
    }
    else {
      ////if it is not a Palindrome String.
      System.out.println([:r +is not a Palindrome String.] );
    }
  }
}
```

```
/* A program to check whether a string or number
is palindrome or not .
A string is called a palindrome string if the rev
erse of that string is the same as the original s
tring. For example, radar, level, etc.

文字列が回文かどうかを判定するプログラム
回文は、上から読んでも下から読んでも同じ文字列に
なる文字列のことです
例：
竹やぶ焼けた
（たけやぶやけた）

実行例1 Execution example1
>Radar is a Palindrome String.

実行例2 Execution example2
>read is not a Palindrome String.
```

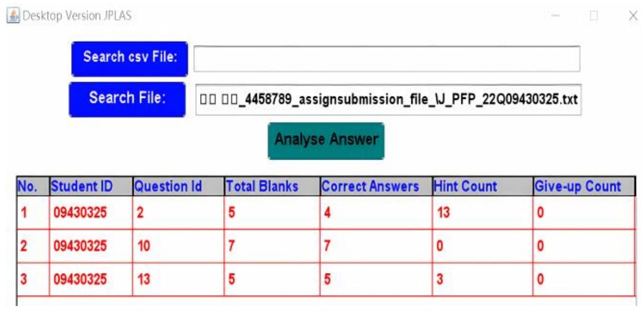
Fig. 5. Answer interface for PFP.

F. *Instance Generation and Solution Steps*

A new instance is generated and is solved through the following steps:

1) Collect the proper source codes of *Java programming* from textbooks.
2) Read a *Java programming* source code file.
3) Run the generation algorithm to generate the input text file.
4) Run the *answer interface generator* with the text file to make the *HTML/CSS/JavaScript* files for the new instance.
5) Upload the generated files to the web server and inform the URL to the students.
6) A student answers to the given instances using a web browser, where the automatic answer marking function will check the correctness of the answers at the browser.
7) The student downloads the answer *text files* at the browser and submits them to the teacher.
8) The teacher runs the *student answer analyzer* with the answer text files and obtains the solving results of the students for each instance.

Fig. 6 displays an example of solving results that is obtained by running the *student answer analyzer*.



□□ □□_4458789_assignsubmission_file_\J_PFP_22Q09430325.txt

No.	Student ID	Question Id	Total Blanks	Correct Answers	Hint Count	Give-up Count
1	09430325	2	5	4	13	0
2	09430325	10	7	7	0	0
3	09430325	13	5	5	3	0

Fig. 6. Solving result for each instance.

IV. Hint Function for Four Problems Types

In this section, we present the *hint function* for GUP, MCP, EFP and PFP of JPLAS.

A. *Basic Concept*

In the hint function in this paper, any hint must be generated automatically from the *input text file* using the corresponding generator. The manual generation of hints will increase the load of a teacher. Since the file contains the correct answer for each question, the hint function will show a part of the characters of the answer as a hint.

Then, only the first character is displayed as the hint for GUP, MCP, and EFP, because they are relatively easy. On the other hand, one or more characters with the number of characters will be displayed for PFP, because PFP is much harder than the others. It has been found that the first character is not sufficient as the hint to solve PFP instances. Besides, to prevent a student from using the hint function facilely, it can be used only after the student clicks the answer button five times and keeps answering it for five minutes as the *thinking time*.

B. *Hint Function for GUP, MCP, EFP*

The *hint function* for GUP, MCP, and EFP will display the first character of the correct answer word for each question where the answer of the student is not correct, when he/she requests it by clicking the corresponding button. Besides, it will show the correct answers to the questions in the instance, if the student cannot reach them after a lot of trials and gives up solving it. Fig. 7 displays the hint for one MCP instance.

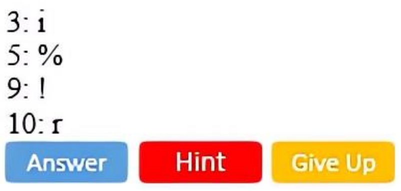


Fig. 7. Hint for MCP.

C. *Hint Function for PFP*

The *hint function* for PFP has two steps. Fig. 8 shows the first step and second step hint for the PFP instance in Fig. 5, respectively. The first step will display the *first character* of the correct answer and the other characters after replaced by ***** to hide them, so that a student can know the number of characters in the answer. The second step will replace ***** by the corresponding character in the correct answer randomly with 30% and display them. Approximately, one-third of the answer can be noticed by the student.

```
Q1: s*****ng****;
Q2: r********* + s************
Q3: r*********.***o*e**a***)
Q4: s** +** a P********* S******
Q5: s** +** n** a P********* S******

Q1: str***ng*h**;
Q2: rev******* + str**********
Q3: r*v*r**S*r.***o*e**ase*)
Q4: str +** a Pal******* Str****
Q5: str +** not a Pal******* Str****
```

Fig. 8. First and second step hint for PFP.

V. Evaluation

In this section, we evaluate the proposed hint function in JPLAS through applications to 38 junior students taking Java programming course in Okayama University. In this application, we selected sample source codes from the course textbook and generated three instances for each of GUP, MCP, EFP, and PFP, and asked the students to solve each problem type within 20min at each of the four classes as quizzes. Any student could use the hint function if necessary.

A. *Results of Individual Problem Types*

First, we discuss the solution performances of the students for individual problem types. Table I shows the number of students who submitted *answer text files*, who used the hint function, and the average correct answer rate among all the students and instances. The average rate for the students who did not use or used the *hint function* are also shown there inside the brackets. It is noted that only one student among 38 did use the *hint function* for GUP.

This table shows that the *hint function* gives the higher correct rate except for MCP. It also shows that as the difficulty level increases, more students used the *hint function* and the correct rate decreased. Particularly, a big gap can be observed between EFP and PFP. One reason may come from the short time of solving PFP instances. Besides, it may be necessary to improve the *hint function* for PFP, and to provide another exercise problem type that can reduce this gap. They will be in future works.

TABLE I: Application Result Summary

problem type	number of students	number of students using hint	ave. correct rate (%) (no hint, hint)
GUP	38	1	91.78 (91.55, 100.00)
MCP	37	8	96.68 (97.18, 94.89)
EFP	34	9	89.82 (89.13, 91.74)
PFP	35	15	39.83 (38.23, 41.96)

B. *Results of Individual Students*

Next, we discuss the solution results of individual students. Fig. 9 shows the average correct answer rate (%) and the number of button clicking for the hint function for GUP, MCP, EFP, and PFP by each student. A total of 33 used the hint function. Here, we note that this number of button clicking is increased by one for each type when the students clicked it, and is increased by two if he/she clicked it at the both steps for PFP.

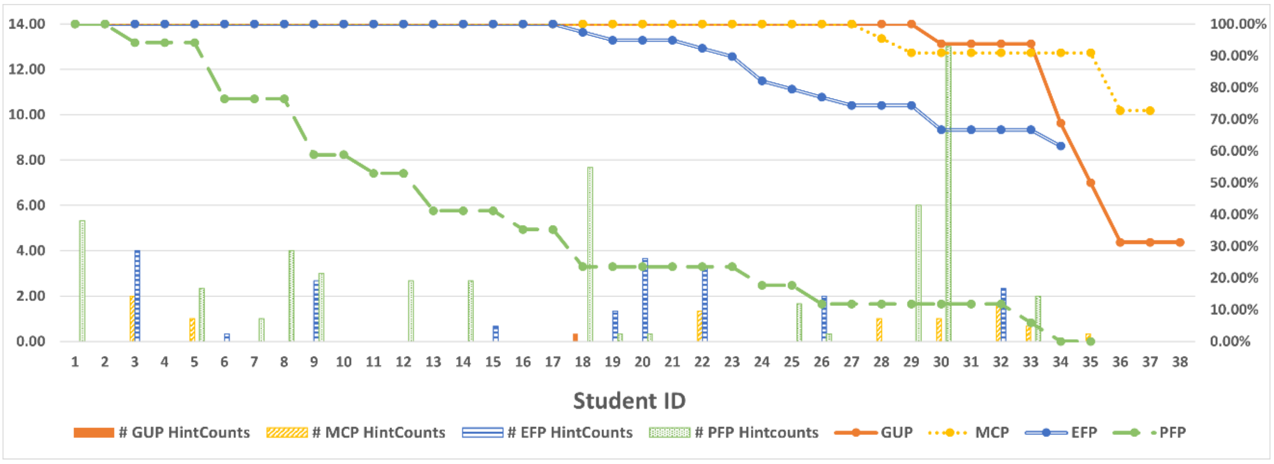


Fig. 9. Solution results of individual students.

In GUP, MCP and EFP, most of the students achieved over the 90% rate. On the other hand, in PFP, more than half of them could not reach 50%. It can be considered that many students cannot complete solving the PFP instances within this limited time, and may be insufficient in programming knowledge and skills.

C. *Results of Students Using Hint Function*

Finally, we discuss the solution results of the students who used the *hint function*. Fig. 10 shows the number of students who used the *hint function* by the range of the correct answer rate. It indicates that some students reached 100% by using the *hint function*, but more students resulted in poor results regardless of the hint function. This result may come from the short time of solving the problems as quizzes. In the hint function, a student can use it only after five minutes have passed as the thinking time since he/she accesses it. Thus, the quiz time 20min was too short for them to freely use the function and solve the questions by referring to the given hints. We have to remedy this undesirable situation by adjusting the *thinking time* from the next application.

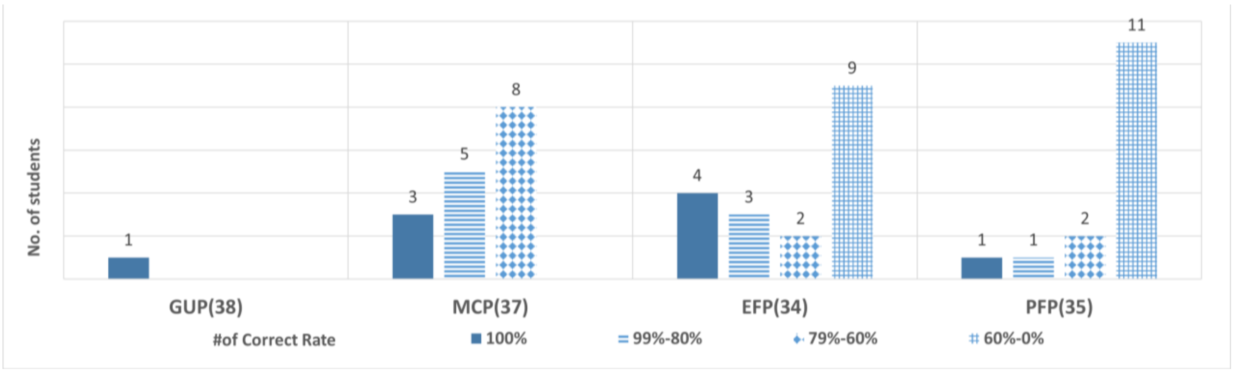


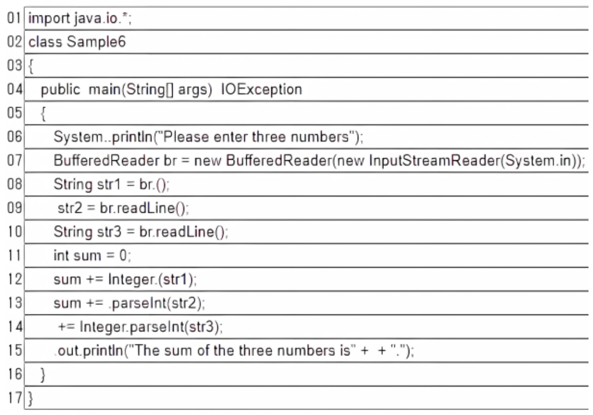
Fig. 10. The number of students by using hints and their correct answer rate.

VI. Hint Function for Code Completion Problem

In this section, we present a *hint function* for *code completion problem (CCP)* by extending the one for PFP. CCP is another type of exercise problems where the correctness of an answer is checked through *string matching*. Our homework assignments of six types of exercise problems in the Java programming course found that the solution result for CCP is the lowest [20]. Therefore, the hint function is important for CCP.

A. Code Completion Problem (CCP)

A question in a CCP instance requests to correct and complete each statement in the given source code that has several blank elements and incorrect ones. Fig. 11 illustrates the answer interface for an example CCP instance. The source code prompts a student to enter three numbers, reads them from the console using a BufferedReader object, converts them from strings to integers, calculates their sum, and prints the result to the console.



```
01 import java.io.*;
02 class Sample6
03 {
04   public  main(String[] args)  IOException
05   {
06     System..println("Please enter three numbers");
07     BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
08     String str1 = br.();
09      str2 = br.readLine();
10     String str3 = br.readLine();
11     int sum = 0;
12     sum += Integer.(str1);
13     sum += .parseInt(str2);
14      += Integer.parseInt(str3);
15     .out.println("The sum of the three numbers is" +  + ".");
16   }
17 }
```

Fig. 11. Answer interface for CCP.

B. Homework Solution Results

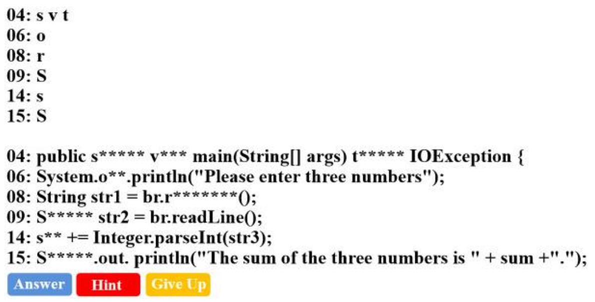
In this Java programming course, we generated a total of 109 instances for GUP, MCP, VTP, EFP, CCP, and PFP, and assigned them to the students as homework. Thus, no time limitation was imposed. Table II shows the number of students who submitted answer files, the average correct answer rate, and the average number of answer submission times for each instance by the students for each problem type. This table indicates that CCP has the lowest correct rate and the second highest number of submissions among the six types of problems, and is most difficult for them. Thus, the hint function will be very important for CCP.

TABLE II: Application Result Summary

problem type	number of instances	number of submitted students	average correct rate (%)	average number 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
total	109	average	97.04	4.09

C. Hint Function for CCP

The *hint function* for CCP consists of two steps like PFP. Fig. 12 shows the first step hint and the second step hint for the CCP instance in Fig. 11, respectively. The first step will display the *first character* of the correct answer for each blanked or incorrect element in the given source code, when he/she clicks the corresponding button to request it. The second step will display the whole statement except the blanked or incorrect element where only the first character is correctly shown, and the remaining characters are replaced by *. Then, a student can determine the position of the blanked or incorrect element in the statement as well as the number of characters.



```
04: s v t
06: o
08: r
09: S
14: s
15: S

04: public s***** v*** main(String[] args) t***** IOException {
06: System.o**.println("Please enter three numbers");
08: String str1 = br.r*******();
09: S***** str2 = br.readLine();
14: s** += Integer.parseInt(str3);
15: S*****.out. println("The sum of the three numbers is " + sum +".");
```

Fig. 12. First and second step hint for CCP.

VII. Conclusion

This paper proposed the hint function for the *grammar-concept understanding problem (GUP), the mistake correction problem (MCP), the element fill-in-blank problem (EFP), and the phrase fill-in-blank problem (PFP) in Java programming learning assistance system (JPLAS)*. It will display the first or more characters of each correct answer word in the JPLAS answer interface. Besides, the use of this function by a student can be traced by a teacher to analyze the performance and problem difficulty. For evaluations, three instances were generated for each of GUP, MCP, EFP, and PFP, and were assigned to junior students taking the Java programming course in Okayama University, Japan. Their answer results confirmed the effectiveness in helping students solve exercise problems in JPLAS, but the thinking time needs to be adjusted for PFP. In future works, we will evaluate the hint function for CCP, provide diverse hint functions depending on students' levels, generate new instances on hard topics for students, and assign them in Java programming courses.

Conflict of Interest

The authors declare no conflict of interest.

Author Contributions

Y. Jing conducted the research and wrote the paper. N. Funabiki and K. Ueda reviewed and finalized the paper. S. T. Aung, X. Lu, and H. H. S. Kyaw collected and analyzed the data. All the authors had approved the final version.

Acknowledgment

We would like to thank the students taking Java programming course in Okayama University to answer the quizzes instances and give us valuable comments. They are inevitable to complete this paper.

References

[1] N. Ishihara, N. Funabiki, M. Kuribayashi, and W. C. Kao, "A software architecture for Java programming learning assistant system," *Int. J. Comput. Soft. Eng.*, vol. 2, no. 1, 2017.

[2] 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, Apr. 2022.

[3] S. T. Aung, N. Funabiki, Y. W. Syaifudin, H. H. S. Kyaw, S. L. Aung, N. K. Dim, and W.-C. Kao, "A proposal of grammar-concept understanding problem in Java programming learning assistant system," *J. Adv. Inform. Tech.*, vol. 12, no. 4, pp. 342–350, Nov. 2021.

[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. Inf. Educ. Technol.*, vol. 12, no. 11, pp. 1158–1163, 2022.

[5] K. K. Zaw, N. Funabiki, and W.-C. Kao, "A proposal of value trace problem for algorithm code reading in Java programming learning assistant system," *Inf. Eng. Express*, vol. 1, no. 3, pp. 9–18, Sep. 2015.

[6] 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. Comput. Sci.*, vol. 44, no. 2, pp. 247–260, May 2017.

[7] 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. Comput. Sci.*, vol. 47, no. 3, pp. 350–359, Aug. 2020.

[8] 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, Mar. 2022.

[9] N. Funabiki, Y. Matsushima, T. Nakanishi, and N. Amano, "A Java programming learning assistant system using test-driven development method," *IAENG Int. J. Comput. Sci.*, vol. 40, no.1, pp. 38–46, Feb. 2013.

[10] J. Yi, U. Z. Ahmed, A. Karkare, S. H. Tan, and A. Roychoudhury, "A feasibility study of using automated program repair for introductory programming assignments," in *Proc. J. FDN. Soft. Eng*, pp. 740–751, Aug. 2017.

[11] S. Marwan, J. Jay Williams, and T. Price, "An evaluation of the impact of automated programming hints on performance and learning," in *Proc. ACM. Int. Conf. Educ. Ree.*, pp. 61–70, Jul. 2019.

[12] T. W. Price, Y. Dong, R. Zhi, B. Paaßen, N. Lytle, V. Catete ́, and T. Barnes, "A comparison of the quality of data-driven programming hint generation algorithms," *Int. J. AI. Educ.*, vol. 29, pp. 368–395, 2019.

[13] A. Rubinstein, N. Parzanchevski, and Y. Tamarov, "In-depth feedback on programming assignments using pattern recognition and real-time hints," in *Proc. ACM. Conf. Inno. Technol. Comput. Sci. Educ.*, pp. 243–244, Jul. 2019.

[14] S. Serth, R. Teusner, and C. Meinel, "Impact of contextual tips for auto-gradable programming exercises in MOOCs," in *Proc. Conf. Learn. @Scale*, pp. 307–310, Jun. 2021.

[15] B. Fein, F. Obermu ̈ller, and G. Fraser, "CATNIP: an automated hint generation tool for Scratch," in *Proc. Conf. Innov. Technol. Comput. Sci. Edu.*, vol. 1, pp. 124–130, Jul. 2022.

[16] Node.js. [Online]. Available: https://nodejs.org/en

[17] Express.js. [Online]. Available: https://expressjs.com/

[18] EJS - Embedded JavaScript templates. [Online]. Available: https://ejs.co/

[19] Docker. [Online]. Available: https://www.docker.com/

[20] X. Lu, N. Funabiki, S. T. Aung, Y. Jing, and S. Yamaguchi, "An implementation of Java programming learning assistant system in university course," in *Proc. ICIET*, March 2023.