An Implementation of Code Validation Function in C Programming Learning Assistant System

Annisa Anggun Puspitasari[1], Nobuo Funabiki[1], Xiqin Lu[2,*], Huiyu Qi[2], Htoo Htoo Sandi Kyaw[3], and Kiyoshi Ueda[4]

[1] Okayama University, Japan; Email: annisanggun@gmail.com (A.A.P.), funabiki@okayama-u.ac.jp (N.F.)
[2] Graduate School of Natural Science and Technology, Okayama University, Japan;
Email: pch55zhl@s.okayama-u.ac.jp (H.Q.)
[3] Division of Advanced Information Technology and Computer Science, Tokyo University of Agriculture and Technology, Tokyo, Japan; Email: htoohtoosk@go.tuat.ac.jp (H.H.S.K.)
[4] College of Engineering, Nihon University, Koriyama, Japan; Email: ueda.kiyoshi@nihon-u.ac.jp (K.U.)
*Correspondence: p06v8z20@s.okayama-u.ac.jp (X.L.)

***Abstract*—In many universities around the world, *C programming* is offered as the first programming course. To help novice students learn on their own, we have developed the *C Programming Learning Assistant System (CPLAS)*. Currently, CPLAS offers simple practice questions at the elementary level, where any question requires a word, a sentence, or a number as the answer and the student answer is checked against the correct answer by *string matching*. However, CPLAS does not cover the problems of writing source codes completely from scratch such that the correctness of each code must be validated automatically. In this paper, we present the implementation of the *code validation function* for validating the answer source code through 1) compiling test, 2) execution test, and 3) output test. Here, the *software test* approach using *test codes* is not adopted, because it might be too difficult for novice students. For evaluations, we applied the proposal to 2, 045 source codes from 43 first-year students at the *C programming* course in Nihon University, Japan. We analyzed the answer results and confirmed the effectiveness of the proposal. In future works, we will introduce this function in the CPLAS platform, which will be used to help students with self-studies.**

***Keywords*—C programming, C *Programming Learning Assistant System* (CPLAS), novice student, self-study, code validation**

I. Introduction

Presently, *C programming* is offered at the first programming courses to undergraduate students in a lot of universities around the world. *C programming* is useful for studying advanced and practical programming languages, such as *Java*, *Python*, and *JavaScript*, which are more suitable and easier to implement practical and large-scale application systems. Besides, it can be used to study fundamental data structure and algorithms, and computer architecture for students in *Information Technology (IT)* or *Computer Science (CS)* departments all of which are essential subjects in the computing curriculum. In [1], Saqib mentioned that knowledge and understanding of computer programming in *C* and *C++* is one of the most fundamental skills for today's students.

To assist self-studies of *C programming* by novice students, we have developed the *C Programming Learning Assistant System (CPLAS)*. Currently, CPLAS provides simple exercise problems at elementary levels, including the *Grammar-Concept Understanding Problem (GUP)* [2], the *Value Trace Problem (VTP)* [3], the *Element Fill-in-Blank Problem (EFP)* [4], the *Code Completion Problem (CCP)* [5], and the *Phrase Fill-in-Blank Problem (PFP)* [6]. In them, an instance consists of a source code, a set of questions and their correct answers. Any question requires a word, a sentence or a number as an answer, which is checked for correctness by *string matching* against the correct answer stored in the system. The common answer interface has been implemented on a web browser [7]. The marking function was implemented by *JavaScript* that runs on the web browser.

The outline and learning goal of each exercise problem are described as follows:

- *GUP* reminds the knowledge and concepts of *reserved words* and common libraries in the given source code, for *grammar* study.
- *VTP* questions the values of important variables and output messages in the source code.
- *EFP* requests to fill in the blank elements in the source code with their originals by understanding the syntax and semantics.
- *CCP* does not show the locations of the blank elements in *EFP*.
- *PFP* requests to fill in the blank phrases that may consist of multiple elements in the source code.

These problems are mainly designed for *code reading* and *code understanding* studies [8, 9]. They do not include the problems of completely writing source codes from scratch that satisfy the specification requirements for

Manuscript received June 11, 2022; revised July 12, 2022; accepted August 7, 2022.

coding study, although the ability to truly master *C programming* and be able to write code from scratch is essential.

In this paper, we present the implementation of the *code validation function* for validating the answer source code from a student to a given assignment for the *Code Writing Problem (CWP)* in *CPLAS*. This function is implemented in *Python*, and validates the correctness of the source code through the three tests: 1) compiling test, 2) execution test and 3) output test. The *compiling test* compiles the source code using *GCC* compiler. The *output test* checks whether the source code correctly outputs the data or messages with the correct format for the given input that are specified by the teacher.

The research question in this paper is: "Can we implement an efficient environment for *CWP* in *CPLAS* using *Python*?". Unlike *CWP* for *Java programming* [10], the *software test* approach using *test codes* is not adopted here, because it can be too difficult for novice students. In the software test approach, the students have to understand the roles of the test code and to write the source code that can be tested by the test code. *Python* offers rich and useful library functions that help to implement the *code validation function* with short and readable codes. This feature will be beneficial for future use and customizations by a lot of programming course teachers.

For the evaluation, we applied the implemented function to 2,045 answer source codes to 50 assignments from 43 first-year students who took the *C programming* course in Nihon University, Japan. The results found that 1, 583 source codes among them successfully passed all of the tests and became correct, where 30 codes failed in the *compiling test*, 8 codes failed in the *execution test* and 424 codes failed in the *output test*. With this implemented function, the teacher can easily perceive the progress or stagnance in *C programming* study of every student. Thus, the effectiveness of the proposal is confirmed.

The implementation of the *code validation function* in this paper intends the use of teachers in *C programming* courses. By using this function, teachers can validate a lot of source codes submitted from students in the courses automatically and check the validation results for each assignment at a glance. However, *CPLAS* has been developed for self-studies of students. Therefore, in future works, we will include this function in the *CPLAS* platform that is under developments using *Node.js* for the web application server.

The rest of this paper is organized as follows: Section II presents the implementation of the *code validation function*. Section III shows the application results to source codes from novice students for *C programming* assignments. Section IV discusses related works in literature. Finally, Section V concludes this paper with future work.

II. Related Works

In this section, we discuss the related works in literature.

Freund *et al.* [11] developed the *Thetis* programming environment that is designed specifically for student use. This system consists of the *C interpreter* and the associated user interface to provide simple and easily understood editing, debugging and visualization capabilities. It is more suitable for students in introductory computer science particularly those for languages like *ANSI C*, assume the level of sophistication that novice students do not possess.

Godefroid *et al.* [12] presented *Directed Automated Random Testing (DART)* for automatically testing software that combines (1) automated extraction of the interface of a program with its external environment using static source-code parsing, (2) automatic generation of a test driver for this interface that performs random testing and (3) dynamic analysis of how the program behaves under random testing and automatic generation of new test inputs. Experiments were conducted to *C source codes*.

Lahtinen *et al.* [13] studied the difficulties in learning basic programming, which include concepts that require understanding larger entities of the program, abstract concepts like pointers and memory handling and a group of topics such as input, output and libraries. They also suggested that students need practical experiences to understand the concepts.

Ihantola *et al.* [14] presented a systematic literature review of the development of automatic assessment tools for programming exercises between 2006 and 2010.

Salleha *et al.* [15] presented preliminary results of research related to programming teaching tools in 45 papers in the *ACM digital database* between 2005 and 2011 with questions: What are the important issues in programming teaching and learning research? What are the methods of the research? What kind of tools involved in programming teaching and learning? What is the level of programming involved? Most of them concerned on techniques and methods in teaching, learning and assessment, and focused on introductory stage.

Dolgopolovas *et al.* [16] presented a case study on how novice engineering students can be motivated to study structured programming and coding in C using game programming in the *App Inventor* environment.

Sobral [17] discussed the choice of the initial programming language to be adopted in the first programming course. It also listed the languages that are currently most widely adopted in the "real world" and in introductory programming courses in higher educations.

III. Implementation of Code Validation Function

In this section, we present the implementation of the *code validation function* for *CPLAS*. This function applies the *compiling test*, the *execution test* and the *output test* sequentially to every *answer source code file* that is stored in the designated folder.

A. Compiling Test

The *compiling test* compiles the source code files by using the *gcc* compiler. However, if syntax errors are detected by the compiler, then this function will report the corresponding message in the *error message file* from the compiler. Otherwise, the next test will be applied.

The current implementation of the function does not assume specific compile options for each assignment, because it should cover only introductory programming assignments. However, the study of various compile options will be necessary for students. For this case, the teacher should prepare the *Makefile* to compile the code with the necessary options. The extension of the function will be in future works.

B. *Execution Test*

The *execution test* runs the execution code that is obtained in the *compiling test*, with the input data specified by the teacher for each assignment if it exists. Then, if some errors are detected by the *Operating System (OS)* while running the code, such as the incorrect input data format, the zero division, the buffer overflow and the infinite loop, this function will report the corresponding message from the *OS*. For the infinite loop, the maximum running time is set at the execution. If the code passes this test, the next test will be applied.

For this test, the teacher prepares the *input file* containing the data or messages to be given to the code and puts it at the specified file path. The teacher is allowed to change the file path, but it is important to note that the *input file* can be omitted when it is not necessary. Here, command line parameters, standard input and file input are supported. When the file input is included in the source code, the teacher requests the students to use the specified file path for the source code of this file input in the assignment.

Likewise, the current implementation does not assume that each assignment has specific options to run the code. To allow them, the teacher should prepare the *script file* to run the code with the necessary options. The extension of the function will be done in future works.

C. *Output Test*

The *output test* checks whether the source code correctly outputs the data or messages for the given input that is specified by the teacher. For this test, the teacher needs to prepare the *output file* that contains the data or messages to be output from the code and the output data or messages will be stored in the *execution result file*. Then, the *Levenshtein distance* between the texts in the two files will be calculated. If the distance is 0, this test regards the source code as the correct one. Otherwise, it will point out the parts of the text in the *execution result file* that are different from the text in the *output file*.

Here, the standard output and the file output are supported. When the file output is included in the source code, the teacher requests the students to use the specified file path in their source codes for this file output in the assignment.

D. *Default File Paths*

In this implementation, we use the following paths to store the necessary files in the file system as the default:

- Answer source code files:
  (F): \\StudentFiles\\(AssignmentGroup)
  \\(AssignmentID)\\(StudentID)\\(StudentID_source
).c
- Input file:
  (F): \\StudentFiles\\(AssignmentGroup)
  \\(AssignmentID)\\inputfile.txt
- Report file:
  (F): \\StudentFiles\\(AssignmentGroup)
  \\(AssignmentID)\\report_(AssignmentID).xlsx
- Output file:
  (F): \\StudentFiles\\(AssignmentGroup)
  \\(AssignmentID)\\(StudentID)
  \\(StudentID)_output.txt
- Compiler error message files:
  (F): \\StudentFiles\\(AssignmentGroup)
  \\(AssignmentID)\\(StudentID)\\(StudentID)_error.t
  xt

It is noted that *(AssignmentGroup)*, *(AssignmentID)* and *(StudentID)* should represent the folder name foreach assignment group, each assignment in the group and each student, respectively. Here, assignments are often categorized into several groups in programming courses, because several programming topics such as *reserved words* and *libraries* are related with each other. *(StudentID)*_source).c should represent the answer source code file of each student that will be validated by the function. These names and the disk drive should be properly selected by the user and be described in the source codes of the function. The user interface to help the descriptions will be done in future works.

E. *Application Example*

Here, we illustrate the application example of the *code validation function* to a source code for the assignment #29 *age sketch* in Table I. This assignment assumes an apartment with four rooms on each of the three floors. The program requests to input the age of the residence of each room from the first room on the first floor using the standard input and to output the age of the residence from the first room on the third floor until the fourth room on the first floor.

1) Source Code File: Fig. 1 shows the example *answer source code file* from a student for this assignment that has several errors and mistakes.

2) Compiling Error Message File: Fig. 2 shows the *compiling error message file* from the *compiling test* to this source code. To pass the *compilation test*, the two syntax errors must be removed, where wfile at line 14 should be replaced by while and; at line 22 should be replaced by +.

3) Input and Output Files: For this assignment, the teacher may prepare the *input file* in Fig. 3 as the input data, and the *output file* in Fig. 4 as the corresponding correct output data.

4) Output Test Result File: Even if the syntax errors in the source code in Fig. 1 are removed, it may cause errors in the *output test*. Fig. 5 shows the *output test result file* from this test. It suggests that the output text at the sixth line from the source code is different from the text at the sixth line in the *output file* and no output text appears at the seventh to ninth lines from the source code, although the *output file* has the texts at the corresponding lines.

```
1   #include<stdio.h>
2   #define NSIZE 3
3   #define MSIZE 4
4   Int main(void){
5   int i, j;
6   int score [ NSIZE ] [ MSIZE ] ;
7   printf("Please enter thr age of the resident\n");
8      for(i=0; i<NSIZE; i++)
9        printf("<< %dth floor >>\n", i+1);
10       for(j=0;j<MSIZE;j++){
11         do{
12             printf("Room %d:", j+1);
13              scanf("%d", &score[i] [j]);
14         }wfile(score[i] [j]<0);
15     }
16  }
17      printf("\n");
18      printf("<< Age sketch of room redident >>\n");
19      for(i=0;j<MSIZE;i++){
20         printf("[%dth floor]",NSIZE-i);
21         for(j=0;j<MSIZE;j++){
22           printf(" (Rm. %d)%d y.o. ",j;1,score[NSIZE-i-1] [j]);
23         }
24        printf("\n");
25     }
26     return 0;
27  }
```

Figure 1. Answer source code file.

```
en9 −2−350.c : In function ' main ' :
en9 −2−350.c : 1 4 : 3 : error : expected  ' while '  before  '
      wfile '
   } w f i l e ( score [ i ] [ j ] <0) ;
       ^~~~~
  en9 −2−350.c : 2 2 : 3 5 : error : expected ' ) '
        before ' ; ' token
  p r i n t f (" (Rm. %d )%d y . o . " , j ; 1 , score [ NSIZE−i −1] [ j
        ] ) ;
                                                          ^
                                                          )
```

Figure 2. Compiling error message fil.

```
1  Please enter the age of the resident << 1 th floor>>
2  Room 1:25
3  Room 2 : 24
4  Room 3 : 23
5  Room 4 : 26
6  << 2 th floor>>
7  Room 1 : 22
8  Room 2 : 25
9  Room 3 : 24
10   Room 2 : 27
11  << 3 th floor>>
12  Room 1 : 25
13  Room 2 : 27
14  Room 3 : 27
15  Room 4 : 23
```

Figure 3. Input file.

```
1  << Age sketch of room resident >>

2  [ 3 th  f l o o r ]  (Rm.  1) 25  y . o .  (Rm.  2) 24  y . o .   (Rm. 3) 23
   y . o .   (Rm. 4) 26 y . o .

3  [ 2 th  f l o o r ]  (Rm.  1) 22  y . o .  (Rm.  2) 25  y . o .   (Rm. 3) 24
   y . o .   (Rm. 4) 27 y . o .

4  [ 1 th  f l o o r ]  (Rm.  1) 25  y . o .  (Rm.  2) 27  y . o .   (Rm. 3) 27
   y . o .   (Rm. 4) 23 y . o .
```

Figure 4. Output file.

```
Output checking  process  using  Levenshtein  distance for
       each l i n e :

Student Output : Line 6 −> << Age sketch of room redident >>

Correct Output : Line 6 −> << Age sketch of room resident >>

Levenshtein  distance  in   Line  6 : 2

Student Output : Line 7 −>

Correct  Output  :  Line  7 −>  [ 3 th  f l o o r ]  (Rm.  1) 25  y .
       o.   (Rm.   2) 24 y .o .(Rm.   3) 23  y . o .  (Rm.   4) 26
       y .o.

Levenshtein  distance  in   Line  7 : 74

Student Output : Line 8 −>
Correct  Output  :  Line  8 −>  [ 2 th  f l o o r ]  (Rm.  1) 22  y .
       o.   (Rm.   2) 25  y . o .    (Rm.   3) 24  y . o .    (Rm.
       4) 27 y . o.

Levenshtein  distance  in   Line  8 : 74

Student Output : Line 9 −>}
Correct Output : Line 9 −> [ 1 th f l o o r ] (Rm. 1) 25 y .
       o.(Rm.   2) 27  y . o .   (Rm.   3) 27  y . o .    (Rm. 4)
       23 y . o
Levenshtein  distance  in   Line  9 : 74
                   Total   Levenshtein distance :   224
```

Figure 5. Output test result file.

Actually, this code has mistakes at lines 18 and 19, where redident at line 18 should be replaced by resident and for (i=0; j<MSIZE; i++) at line 19 should be replaced by for (i=0; i<NSIZE; i++). Then, the source code will output the correct one.

IV. Evaluation

In this section, we evaluate the *code verification function* of 50 assignments from *C programming* course by applying the source code of 2,045 answers from 43 first-year students at Nihon University in Japan.

A. Programming Assignments

Table I shows the programming topic, the input data type and the output data type for each assignment in this application. Basically, elementary topics are selected for programming assignments to first-year undergraduate students.

B. Application Results

Table II shows the number of students who submitted answer source codes, the total *CPU time* to run the function, the number of codes that passed the three tests, the number of codes that failed at the *compiling test*, the number of codes that failed at the *execution test* and the number of codes that failed at the *output test* for each assignment. Table III shows the *PC* specification to run the function.

Table II indicates that among the 2,045 source codes, 1,583 (77.41%) codes were correct, 30 (1.47%) codes have syntax errors, 8 (0.39%) codes have infinite loops, and 424 (20.73%) codes have output errors. The teacher can easily figure out the progress or stagnance in *C programming* study of each student. Thus, the effectiveness of the implemented function is confirmed.

TABLE I. Assignment Overview

ID#	topic	input	output
1	max number	std	std
2	decending order	no	std
3	even odd	std	std
4	larger smaller	std	std
5	max min	std	std
6	apples oranges	std	std
7	while loop	no	std
8	summation	no	std
9	summation 2	std	std
10	student scores	std	std
11	area of triangle	std	std
12	even number	std	std
13	month	std	std
14	asterisks	std	std
15	distance of two points	std	std
16	birthday	std	std
17	max value	std	std
18	max value in group	std	std
19	max min in group	std	std
20	array total value	no	std
21	array total value 2	std	std
22	average max min array	std	std
23	finding value in array	std	std
24	max score	std	std
25	calculation	std	std
26	average max min array 2	std	std
27	vector	std	std
28	student scores 2	std	std
29	age sketch	std	std
30	OX	no	std
31	subject scores	std	std
32	split letters	std	std
33	arrange letters	std	std
34	reverse string	std	std
35	prefecture	std	std
36	Nichidai Taro	no	std
37	student list	std	std
38	student scores 3	std	std
39	student scores 4	std	std
40	outputting input data	std	std
41	average calculation	std	std
42	max calculation	std	std
43	max calculation 2	std	std
44	max calculation 3	std	std
45	board games	std	std
46	board games 2	std	std
47	board games 3	std	std
48	board games 4	std	std
49	board games 5	std	std
50	board games 6	std	std

TABLE II. Application Results

ID	# of students	CPU time (sec)	correct code	comp. error	exec. error	output error
1	42	22.49	34	1	0	7
2	42	12.77	37	0	0	5
3	43	13.93	37	0	0	6
4	43	14.50	32	0	0	11
5	42	12.30	39	0	0	3
6	40	11.90	27	1	0	12
7	43	13.22	42	0	0	1
8	43	12.99	43	0	0	0
9	43	18.97	36	0	0	7
10	42	13.33	35	0	0	7
11	42	24.00	26	0	0	16
12	41	11.26	37	1	0	3
13	42	19.15	41	0	0	1
14	42	11.54	39	1	0	2
15	42	11.32	36	2	0	4
16	41	18.99	38	0	0	3
17	40	11.37	40	0	0	0
18	40	11.67	36	0	0	4
19	40	19.86	18	0	0	22
20	42	13.76	40	0	0	2
21	42	11.75	34	1	0	7
22	41	12.31	32	1	0	8
23	40	12.89	28	1	0	11
24	42	17.46	38	1	0	3
25	41	12.56	39	2	0	0
26	39	19.22	31	0	0	8
27	38	13.28	33	0	0	5
35	41	13.28	20	1	1	19
36	43	13.13	41	0	0	2
37	41	11.80	38	0	0	3
38	40	11.61	19	2	0	19
39	39	14.95	20	0	0	19
40	40	17.47	23	2	0	15
41	39	17.56	19	1	0	19
42	38	14.24	12	1	0	25
43	38	14.41	2	1	0	35
44	38	13.30	20	2	0	16
45	42	11.36	37	1	1	3
46	41	12.73	33	0	0	8
47	41	21.82	34	1	1	5
48	40	21.38	31	1	0	8
49	38	20.59	29	0	1	8
50	38	20.47	26	1	1	10
total	2045	741.21	1583	30	8	424

TABLE III. PC Specification

CPU	Intel(R) Core(TM) i5-6200U 2.3GHz
memory	12GB
OS	Windows 10 Pro 64-bit
compiler	gcc 8.1.0
item	specification

V. Conclusion

This paper presented the implementation of the *code validation function* for validating the answer source code of a *C programming* assignment through 1) compiling test, 2) execution test and 3) output test. The evaluation results of 2, 045 source codes from 43 first-year students at the *C programming* course in Nihon University, Japan, confirmed the effectiveness of the function.

In future works, we will generate new instances on other topics in *C programming* and apply them to novice students in various universities and departments. Moreover, we will add new features of testing code implementations such as names and data types of variables and functions and adopted grammars. Furthermore, we will implement the program for automatic input/output file generations to help teachers, the user interface for changing the usage setup depending on the requirements from the users, and import the function into the *CPLAS* platform for self-studies by novice students.

Conflict of Interest

The authors declare no conflict of interest.

Author Contributions

A. Puspitasari mainly conducted the research and wrote the paper. N. Funabiki and K. Ueda reviewed and finalized the paper. X. Lu and H. Qi analyzed the data. H. H. S. Kyaw collected the source codes. All the authors had approved the final version.

Acknowledgment

We would like to thank the students of Nihon University in Japan, who answered the 50 assignments and gave us valuable opinions and suggestions. They are inevitable to complete this paper.

References

[1] M. Saqib. Why earn C as first programming language? [Online]. Available: https://www.mycplus.com/featured-articles/why-learn-c-as-first-programming-language/

[2] X. Lu, S. T. Aung, H. H. S. Kyaw, *et al.*, "A study of grammar-concept understanding problem for C programming learning," in *Proc. LifeTech*, March 2021, pp. 162–165.

[3] X. Lu, N. Funabiki, H. H. S. Kyaw, *et al.*, "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] H. H. S. Kyaw, N. Funabiki, S. L. Aung, *et al.*, "A study of element fill-in-blank problems for C programming learning assistant system," *Int. J. Inform. Edu. Tech. (IJIET)*, vol. 11, no. 6, pp. 255–261, 2021.

[5] H. H. S. Kyaw, E. E. Htet, N. Funabiki, *et al.*, "A code completion problem in C programming learning assistant system," in *Proc. ICIET*, March 2021, pp. 34–40.

[6] X. Lu, S. Chen, N. Funabiki, *et al.*, "A proposal of phrase fill-in-blank problem for learning recursive function in C programming," in *Proc. LifeTech*, March 2022, pp. 127–128.

[7] N. Funabiki, H. Masaoka, N. Ishihara, *et al.*, "Offline answering function for fill-in-blank problems in Java programming learning assistant system," in *Proc. ICCE-TW*, May 2016, pp. 324–325.

[8] T. Busjahn and C. Schulte, "The use of code reading in teaching programming," in *Proc. Koli Calling*, 2013, pp. 3–11.

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

[10] N. Funabiki, Y. Matsushima, T. Nakanishi, *et al.*, "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.

[11] S. N. Freund and E. S. Roberts, "Thetis: An ANSI C programming environment designed for introductory use," *SIGCSE Bull.*, vol. 28, no. 1, pp. 300–304, March 1996.

[12] P. Godefroid, N. Klarlund, and K. Sen, "DART: Directed automated random testing," in *Proc. PLDI*, June 2005, pp. 213–223.

[13] P. Vostinar, "Interactive course for JavaScript in LMS Moodle," in *Proc. ICETA*, 2019, pp. 810–815.

[14] P. Ihantola, T. Ahoniemi, V. Karavirta, *et al.*, "Review of recent systems for automatic assessment of programming assignments," in *Proc. Koli Calling*, Oct. 2010, pp. 86–93.

[15] S. M. Salleha, Z. Shukura, and H. M. Judi, "Analysis of research in programming teaching tools: An initial review," *Procedia Soc. Behavi. Sci.*, vol. 103, pp. 127–135, 2013.

[16] V. Dolgopolovas, T. Jevsikova, and V. Dagiene, "From Android games to coding in C – An approach to motivate novice engineering students to learn programming: A case study," *Comput. Appl. Eng. Educ.*, vol. 26, pp. 75–90, 2018.

[17] S. Sobral, "CS1: C, Java or Python? Tips for a conscious choice," in *Proc. Int. Conf. Edu. Res. Innov.*, Nov. 2019.

Annisa Anggun Puspitasari received the B.E. degree in telecommunication engineering from Politeknik Elektronika Negeri Surabaya (PENS), Indonesia, in 2021. She is currently an adjunct researcher at Okayama University, Japan. Her research interests include educational technology and wireless communication systems.

Nobuo Funabiki received the B.S. and Ph.D. degrees in mathematical engineering and information physics from the University of Tokyo, Japan, in 1984 and 1993, respectively. He received the M.S. degree in electrical engineering from Case Western Reserve University, USA, in 1991. From 1984 to 1994, he was with Sumitomo Metal Industries, Ltd., Japan. In 1994, he joined the Department of Information and Computer Sciences at Osaka University, Japan, as an assistant professor, and became an associate professor in 1995. In 2001, he moved to the Department of Communication Network Engineering at Okayama University as a professor. His research interests include computer networks, optimization algorithms, educational technology, and Web technology. He is a member of IEEE, IEICE, and IPSJ.

Xiqin Lu received the B.S. degree in electronic information engineering from Hubei University of Economics, China, in 2017, and received the M.S degree in electronic information systems from Okayama University, Japan, in 2021, respectively. She is currently a Ph.D. student in Graduate School of Natural Science and Technology, Okayama University, Japan. She received the OU Fellowship in 2021. Her research interests include educational technology.

Huiyu Qi received the B.A. degree in information management and information system from Dalian University of Foreign Languages, China, in 2021.She is currently a master student in Graduate School of Natural Science and Technology, Okayama University, Japan. Her research interests include educational technology.

Htoo Htoo Sandi Kyaw received the B.E. and M.E. degrees in information science and technology from University of Technology (Yatanarpon Cyber City), Myamar, in 2015 and 2018, and Ph.D. in information communication engineering from Okayama University, Japan, in 2021, respectively. She is currently an assistant professor in Division of Advanced Information Technology and Computer Science, Tokyo University of Agriculture and Technology, Koganei, Japan. Her research interests include educational technology and web application systems. She is a member of IEICE.

Kiyoshi Ueda received the B.E. and M.E. degrees in electrical engineering from Keio University, Japan, in 1987 and 1989, respectively. He received the Ph.D. degree in information science and electrical engineering from Kyushu University, Japan, in 2010. He was with NTT from 1989 to 2014, where he studied digital switching software, especially distributed switching node management software. He is presently a professor at College of Engineering, Nihon University from 2014. Prof. Ueda is a member of IEEE, IEICE and IPSJ.