ASTES

Advances in Science, Technology and Engineering Systems Journal
Vol. 7, No. 1, 14-26 (2022)
www.astesj.com
Special Issue on Innovation in Computing, Engineering Science & Technology

ASTES Journal
ISSN: 2415-6698

Value Trace Problems for Code Reading Study in C Programming

Xiqin Lu[1], Nobuo Funabiki[*,1], Htoo Htoo Sandi Kyaw[1], Ei Ei Htet[1], Shune Lae Aung[2], Nem Khan Dim[2]

[1]*Department of Electrical and Engineering, Okayama University, Okayama, 700-8530, Japan*

[2]*Department of Computer Studies, University of Yangon, Yangon, 11041, Myanmar*

ARTICLE INFO

Article history:
Received: 05 July, 2021
Accepted: 02 January, 2022
Online: 13 January, 2022

Keywords:
C programming
Value trace problem
Code reading
Self-study
Grammar concept
Algorithm
Pointer

ABSTRACT

C programming is taught in a lot of universities across the world as the first computer programming language. Then, for novice students, it is important to read many simple C source codes and understand their behaviors to be familiar to the programming paradigm. Unfortunately, effective tools to support independent code reading study at home have not been well designed. Heretofore, we have proposed the value trace problem (VTP) for Java programming. A VTP instance consists of one source code, several questions, and the correct answers to them. Each question asks the value of a critical variable or output message in the source code. The correctness of any student answer is checked instantly by string matching at the answer interface for self-study. In this paper, we present the value trace problem (VTP) for code reading self-study of C programming. 42 VTP instances are generated using simple C source codes on basic grammar concepts and fundamental data structures & algorithms in textbooks and websites. In addition, for hard instances on pointer and algorithms, the devices of hints, multiple choice questions, and references are provided to improve their solution performances. For evaluations, we requested 49 *undergraduate students in Japan, China, and Myanmar to independently solve them at home. Their average correct answer rate reached* 94.29%*, where our devices for hard instances improved it by* 33.26%*. Thus, the effectiveness of our proposal is confirmed in motivating self-study of C programming to novice students.*

1 Introduction

Presently, *C programming* is widely taught in many universities across the world as the first computer programming language. In addition to *information technology (IT)* departments, many departments including science, agriculture and mechanical/electrical engineering are teaching *C programming* courses. Actually, the study of *C programming* can offer basic knowledge to study more advanced and practical languages such as *Java*, *JavaScript*, and *Python*. Besides, students in IT departments should study *C programming* in parallel with computer architecture, because they can learn accesses to memories or registers through it. Furthermore, *C* is still the third most popular programming language, despite the long time passed from the appearance. Then, for such novice students, it is important to read and understand a lot of simple source codes to be familiar to the *C programming* structure. Unfortunately, effective tools to support independent *code reading self-study* at home have not been well designed within our knowledge.

Heretofore, we have proposed and implemented the *Java programming learning assistant system (JPLAS)* for *Java programming* study [1]. As the objected-oriented programming language, *Java* is now widely used in IT societies. For assisting *Java programming* studies at different learning levels, JPLAS provides several types of *exercise problems* with different difficulties. For any problem type, the student answer to a question is marked automatically at the system to support programming self-studies.

Among them, the *value trace problem (VTP)* is presented for novice students to study basic concepts of programming grammar and skills through *code reading study* [2]. A VTP instance consists of one source code, several questions, and the correct answer strings to them. Each question asks the value of a critical variable or an important output in the given code. The correctness of any student answer is marked through *string matching* with the correct answer string at the answer interface using the web browser instantly.

The manual collection of C source codes and the manual selection of the variables or output messages from each source code for questions can be the limitation of *VTP*. When a teacher uses *VTP* in the programming course, he/she needs to carefully collect the

[*]Corresponding Author: Nobuo Funabiki, 3-1-1 Tsushimanaka, Okayama University, funabiki@okayama-u.ac.jp

source codes and select the variables or messages with their timing whose actual values or contents should be traced in the questions, so that they can be synchronized with the course progress. One way to generate VTP instances by a teacher efficiently is the use of sample source codes that will be provided with the textbook in the course. Usually, the teacher can download them from the website. Then, he/she can select the variables or output messages that are closely related to the teaching topics in each class of the course. The impossibility of practicing source code writing by a student will be another limitation of *VTP*. It should be offered by other problem types such as the *code writing problem* [3].

In this paper, we present the *value trace problem (VTP)* for *code reading self-study* of *C programming* by extending our previous works for Java programming. To assist the study of novice students, 42 simple C source codes on *basic grammar concepts* and *fundamental data structures & algorithms* are collected from textbooks and websites. Here, *fundamental data structures & algorithms* are often taught using *C* programs in IT departments. Then, by analyzing critical variables/messages and adding necessary standard output statements in them, the corresponding VTP instances are generated manually.

The memory management using *pointer* in *C programming* can be a hard topic for novice students to study, although it is the remarkable feature in implementing fast and efficient programs. Besides, some fundamental algorithms can be difficult, because novice students are often not familiar to them. Therefore, we additionally provide the devices of hints, multiple choice questions, and references to improve the solving performances by the students for the hard VTP instances on *pointer* and *algorithms*.

For evaluations of the proposal, we asked 49 students in Japan, China, and Myanmar to independently solve the generated 42 VTP instances at home. Then, the results showed that the average correct answer rate among them reached 94.29% that is sufficiently high, where the devices for hard instances improved it by 33.26%. Thus, the effectiveness of our proposal is confirmed in motivating self-study of *C programming* by novice students [4]–[10].

VTP can be applied for studying other programming languages such as *JavaScript* and *Python* in addition to *Java* in a straightforward way. Besides, *VTP* can be used for studying a natural language such as English or Japanese as the potential application. For natural language study, the instance may give a sentence or a paragraph and ask the appearing keywords in the questions. Here, the variable for programming study becomes the key concept in the sentence or paragraph for natural language study, and the value becomes the keyword.

The remaining part of this paper is organized as follows: Section 2 discusses related works in literature. Section 3 presents the *value trace problem (VTP)* for *C programming*. Section 4 presents devices to improve the solving performances for hard instances on *pointer* and *algorithms*. Section 5 shows their evaluation results. Finally, Section 6 concludes this paper with future works.

2 Related Works in Literature

In this section, we introduce related works in literature. Some papers presented programming study tools, and some discussed the importance of code reading in programming study.

2.1 Programming Study Tools

In [11], the author reviewed some tools to support teaching and learning of programming, and categorized them into four groups: 1) tools that include simple reduced development environments, 2) example-based environments, 3) tools that are based on visualizations and animations, and 4) simulation environments. Our proposal can be categorized to 2).

In [12], the author presented a tool called *QuizPACK*, which is similar to the *value trace problem (VTP)*, and showed that this tool significantly improved the knowledge of semantics. A question in *QuizPACK* asks the value of a particular expression (usually a variable) in a fragment of a program. A teacher needs to prepare a set of problems using source code fragments and expressions to be questioned manually. If compared with *VTP* in this paper, the advantage of *QuizPACK* is that the constants related to the questioned variables in the code and their correct answers can be generated dynamically so that different students can solve the same question with the different correct answers. On the other hand, *VTP* fixes the constants and the correct answers. The disadvantage of *QuizPACK* is that it can only ask the ending values of the variables in the code fragment. It cannot ask the values at the different timing of the program. Thus, *QuizPACK* is not suitable for studying *algorithms*, where different algorithms can give the same variable values.

In [13], the author presented a problem-solving environment named *LECGO (Learning Environment for programming using C using Geometrical Objects)* for learning *C programming* by beginners. *LECGO* emphasized: (a) multiple external representations in student learning, (b) motivation through performing problem-solving activities from the familiar and meaningful context, (c) the active participation of students by using hands-on experience, (d) appropriate feedback to aid self-corrections, and (e) holistic, activity-based, multi-media, multi-representational and multi-layered content for learning basic concepts.

In [14], the author presented *Gidget*, which is a game such that the eponymous robot protagonist is cast as a fallible character that blames itself for not being able to correctly write code to complete its missions. Players of *Gidget* can learn programming by debugging the problematic code.

In [15], the author presented a game-based learning support environment for novice students to learn programming. This environment exploits game composition tasks to make the elementary programming more intuitive to be learnt, and comprises concept visualization techniques, to allow the students to learn key concepts in programming through game object manipulations.

In [16], the author proposed the *web-based programming assisted system for cooperation (WPASC)* for facilitating cooperative programming learning, and investigated cooperative programming learning behaviors of students and its relationships with learning performances.

In [17], the author presented systematic literature review results on assessment tools for programming assignments, to help instructors make their selections in programming courses. They identified three specialties in assessment tools, namely, *contests*, *quizzes*, and *software testing*. In *contests*, a tool compiles the source code of a

student, runs it with a set of test cases for the input, and informs whether the code is accepted or not. In *quizzes*, it gives a set of questions, where for each question, a student enters a fragment of a code in the interface of the tool. In *software testing*, it checks the correctness of the source code of a student by comparing its output with the output of the model code, or by running the test code against the source code. VTP in this paper is different from any of them.

In [18], the author presented a system that combines the dual objectives of automated grading and program repairing for introductory programming courses called *GradeIT*. It grades the submitted source code with the number of passed test cases, the inverse of time spent to solve, and the fraction of successful compilations. For the last one, the compilation score is defined by 1 – (number of compilation errors in the code) / (maximum number of compilation errors among students). For repairing, it uses simple re-writing rules to fix simple but frequent compile time errors.

In [19], the author proposed a plug-in system to *Moodle* called *LAPLE (Learning Analytics in Programming Language Education)* to provide a learning dashboard to capture the behaviors of the students in the classroom and identify the difficulties faced by different students looking at different knowledge. *LAPLE* asks the students to write full source codes, and collects and analyzes the compiling logs of the students every five minutes to yield real-time visualizations for feedbacks during the class. The authors analyzed the distribution of the classified 36 error types, and encouraged the students to pay attentions to the frequent error types. *LAPLE* is designed for use at programming classes, while *VTP* is for use at homes for self-studies.

2.2 Code Reading in Programming Study

In [20], the author concluded that through literature reviews and interviews, *code reading* is connected to comprehending programs and algorithms, or algorithmic ideas, as well as details, and is needed in many aspects of learning programming, but at the same time, there is not much knowledge about the reading and comprehension process of learners. They claimed that a possible means to foster programming learning is to teach code reading directly, including reading strategies. The value trace problem in this paper can be one way to achieve it for *C programming* study, which supports the novelty of this paper.

In [21], the author concentrated on finding out whether reading programs before writing programs is more efficient for students to learn programming. He asked software developers and found that computer science graduates should understand existing codes, since most developers need to maintain and extend existing systems. He concluded that if students get through the syntax of programming more quickly by conducting code reading exercises, class time could be spent on issues such as the efficiency and readability of software. Code reading problems are important for beginners to learn a new programming language.

In [22], the author concluded that programming learning should focus on algorithmic thinking, rather than programming language features and skills. Programming learning could help improve problem solving skills through algorithmic thinking. The generated VTP instances in this paper cover important algorithmic topics such as stack, queue, and sorting algorithms, which will be helpful for developing algorithmic thinking.

In [23], the author indicated that source code reading behaviors are important in teaching programming skills as well as designing IDEs and programming languages. A source code is a different type of text than a natural-language text. It is highly formal and structured, and has a very limited vocabulary of keywords, operators, and separators, yet, tremendous combination possibilities for literals and identifiers. They recorded programmers' eye movements when reading short source codes. The results show that most attention is oriented towards understanding of identifiers, operators, keywords, and literals, relatively little reading time is spent on separators.

3 Value Trace Problem for C programming

In this section, we present the *value trace problem (VTP)* for *C programming*.

3.1 Definition of Value Trace Problem

The mission of *VTP* is to let students trace the value of a critical variable or an important output in the given source code by carefully reading and understanding its behaviors. To generate a VTP instance, one *source code*, a set of *questions* with the *answer forms*, and the *correct answer strings* to them need to be prepared. A question requests the current value of the important variable or message in the source code that is specified by the teacher. Therefore, the source code should contain the *standard output* statements that correspond to them, so that the students can easily find which values are asked in the questions.

3.2 Design Goals of Value Trace Problem

VTP for *C programming* have the following design goals:

1) A variety of C source codes are given to beginners for *code reading study*.

2) A student can solve the questions in a VTP instance by reading the given source code and understanding its behavior carefully.

3) Any answer from a student will be checked through *string matching* at the interface automatically, and the result will be returned to the student instantly.

4) A teacher can choose proper source codes in terms of contents and difficulty levels for the VTP instance that will be assigned at each class in the programming course.

3.3 VTP Instance Generation Procedure

A new VTP instance can be generated by the following procedure:

1. A proper *source code* for studying a *basic grammar concept* or a *fundamental data structure/algorithm* is selected.

2. The important *variables* or *output messages* to be traced in the source code are selected.

3. The corresponding *standard output statements* are added in the code.

4. The questions of asking the values/messages at the standard outputs and their *correct answers* are prepared.

5. The source code, the corresponding questions, and the correct answers are put together into one *input text file*.

6. The program in [4] is executed with the *input text file* to generate the *CSS/HTML/JavaScript files* for the *answer interface* on the web browser.

7. The generated VTP instance is registered as the assignment to students.

3.4 Answer Interface

The *answer interface* using the web browser in [4] is adopted for the students to solve the generated VTP instances in this paper. The necessary functions such as *string matching* for answer marking and the answer data storage for submissions are implemented in *JavaScript* so that this interface can be used without the Internet connection.

To use this answer interface, the correct answers to the questions need to be distributed to the students, so that their answers can be marked instantly at offline. Then, to prevent the students from looking at the correct answers before solving the questions, the hash values of them are taken using *SHA256* before the distributions. Actually, each correct answer is concatenated with the assignment and problem IDs before hashed, so that the same correct answers for the different questions are converted to the different hash values. It can avoid knowing the correct answer from the same hash value for a different question. At the marking, any answer from a student is concatenated and hashed using *SHA256* for *string matching* with the hashed correct answer.

3.5 Example VTP Instances

Two VTP instances generated in this paper are illustrated here as the typical examples for the *variable trace study* and the *pointer trace study* in *C programming*.

3.5.1 VTP Instance for Variable Trace Study

Figure 1 shows the *input text file* for the *if in for-loop* instance at *ID=12* in Table 1. This file contains the source code, the questions with four answer forms, and their correct answers, which must be done manually. Each correct answer is separated by ",". This instance asks to trace the value of *num* and the logic implemented by two *if*. A student needs to read and understand the source code, and then fill in the forms with correct answers.

```
int main(void){
   int max = 9;
   int n;
   for (n = 2; n <= max; n++){
      if (n == max){
         printf("%d is prime number \n", max);
      }
      else if(max % n == 0){
         printf("%d is not prime \n", max);
         break;
      }
   }
   return 0;
}

What is the value of max? _1_
What is the output? _2_ is _3_ _4_

9,9,not,prime
```

Figure 1: Input text file for VTP instance at *ID=12*.

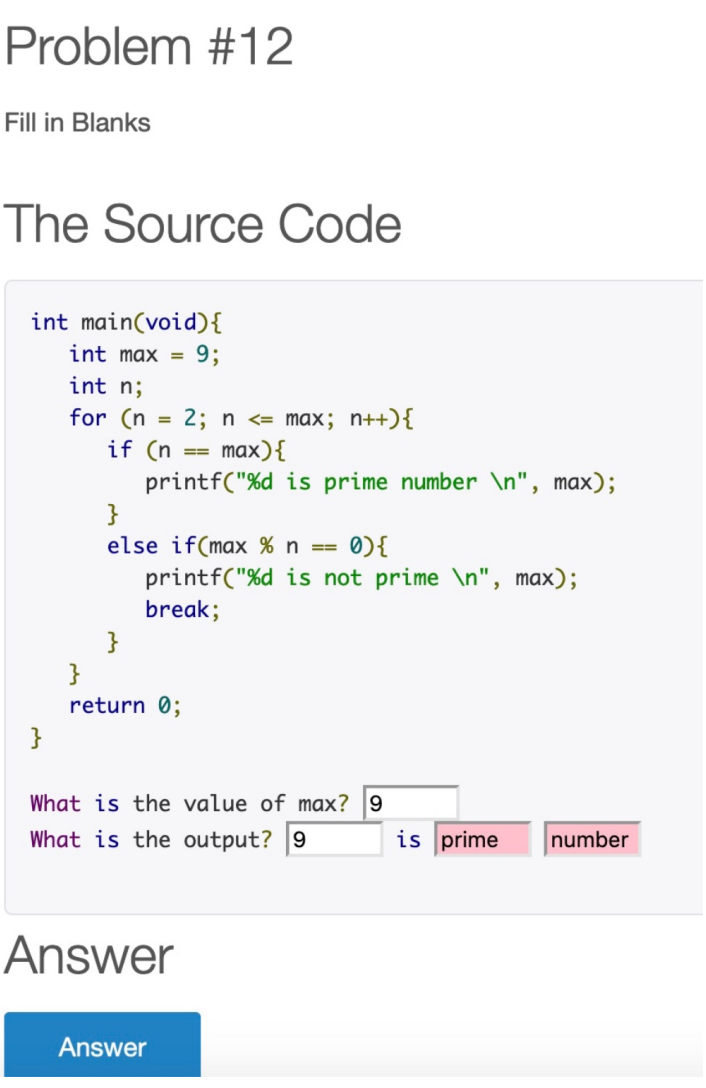

Figure 2: Answer interface for VTP instance at *ID=12*.

```
int main(void)
{
   int test[5] = {80,60,55,22,75};

   printf("test[0] is %d \n", test[0]);
   printf("test[1] is %d \n", test[1]);
   printf("The address of test[0] is %p \n", &
       test[0]);
   printf("test is %p \n", test);
   printf("The address of test[1] is %p \n", &
       test[1]);
   printf("That is *test is %d \n", *test);
   return 0;
}

hint:The address of test[0] is 0028FF20
     1 word 4 bit in this system

What is the output of this program?
test[0] is  _1_
test[1] is  _2_
The address of test[0] is  _3_
test is  _4_
The address of test[1] is  _5_
That is *test  _6_

80,60,0028FF20,0028FF20,0028FF24,80
```

Figure 3: Input text file for VTP instance at *ID=9*.

Figure 2 illustrates the answer interface on a web browser for this VTP instance. After a student fills in the answer forms, he/she should click the blue button "Answer". If the student answer is incorrect, the form color will be *red*. For the correct one, it becomes *white*. Using this interface, a student can fix the mistakes and submit the answers instantly, until all the answers become correct.

3.5.2 *VTP Instance for Pointer Trace Study*

Figure 3 shows the *input text file* for the *memory address with pointer for integer array* instance at *ID=9* in Table 1. This instance asks to trace the memory address or the value given by *pointer* including the one for the array *test*. Figure 4 depicts the answer interface for this sample VTP instance.

3.6 *Generations of Value Trace Problem Instances*

Now, we present the generated 42 *value trace problem (VTP)* instances with 586 answer forms for studying *basic grammar concepts* and *fundamental data structures & algorithms* in *C programming*.

Table 1 shows the topic, the number of lines (#of lines) in the source code, the number of questions, and the number of answer forms in each VTP instance. Here, to encourage novice students solving all of them without giving up halfway, we took the following stances at their generations:

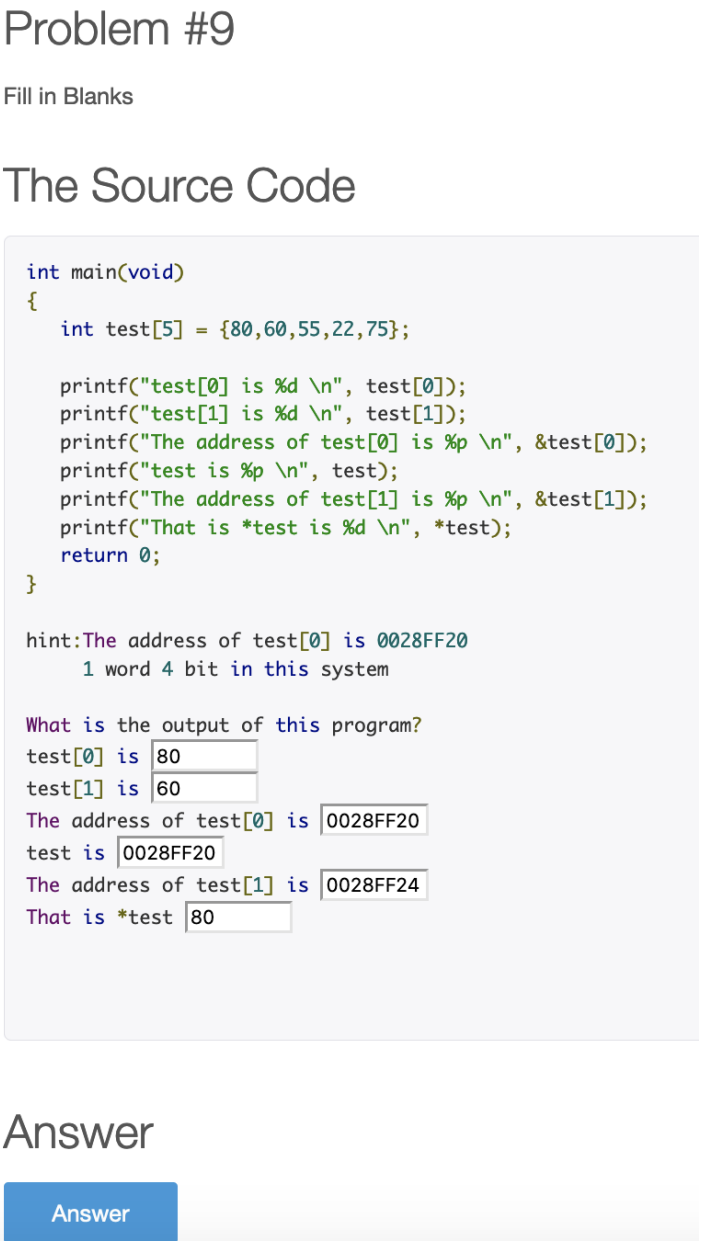

Figure 4: Answer interface for VTP instance at *ID=9*.

1. A short source code with less than 30 lines is basically selected for each instance. Actually, the average number of lines among the source codes becomes 24.5. However, source codes for *fundamental data structures & algorithms* are longer due to their natures.

2. The number of questions is limited to at most four for each instance. Actually, the average number of questions becomes 1.26.

3. The difficulty level of each instance is gradually increased from the first to the last roughly, by following *C programming* textbooks.

3.6.1 VTP Instances for Basic Grammar Concepts

For studying *basic grammar concepts*, 26 VTP instances were generated using the C source codes in the websites [5]-[7]. The selected grammar concepts are essential in fundamental *C programming*.

3.6.2 VTP Instances for Fundamental Data Structures & Algorithms

To study *fundamental data structures & algorithms* of C programming, 16 VTP instances were generated using the source codes in the textbook [8]. They can be hard subjects for novice students in IT departments. To let them read the implemented algorithm procedure in each source code step-by-step and understand the behaviors, the hallway actual values of the important variables in the code are asked in the questions. For example, the actual values of the array variables to store the sorted data and the variable for *pivot* at each iteration are asked in the VTP instance for *quick sort*. These questions can also avoid answering them without correctly understanding the algorithm implementation in the source code.

4 Devices for Hard VTP Instances

In this section, we present our devices to improve the solution performances for hard VTP instances on *pointer* and *algorithms*.

4.1 Hard VTP Instances

In our preliminary application of our generated VTP instances to limited students, the instances at *ID=22*, *ID=28*, *ID=31*, *ID=38*, and *ID=41* in Table 1 showed low correct answer rates compared with the others. These results suggest that not a few students do not understand the topics of *memory address*, *linked list*, *reverse polish notation*, *quick sort*, and *data size of pointer* well. They are related to memory managements by *pointer* or *algorithms*.

To improve the comprehensions on these topics and the solution performances of students, the following devices are presented in this paper:

1) We add *tips* in questions on pointers.
2) We give *hints* in the answer interface on memory addresses.
3) We prepare *multiple choice questions* on linked lists.
4) We add *links to reference websites* in the answer interface on algorithms.

4.2 Tips in Questions

On *tips* for pointers, Figures 5 and 6 show the source code and the questions for the VTP instance at *ID=41*. In the questions, the size of one word in this system is described by *(64-bit size system)*, and the targeting data in each question is described by *Data size of normal variables*, *Data size of pointer*, and *Data size of pointer reference*.

Table 1: Generated 42 VTP instances.

ID	topic	#of lines	#of questions	#of forms
1	two-dimensional array	18	1	12
2	max function	7	1	5
3	arithmetic	6	1	6
4	data type	12	1	16
5	for-loop	10	1	1
6	for-loop with character	11	1	9
7	function call	12	1	10
8	function for swap	21	1	4
9	memory address with pointer for integer array	11	1	56
10	car structure	12	1	2
11	output in for-loop	9	1	8
12	if in for-loop	15	2	4
13	book structure	14	1	3
14	pointer for one-D array	8	1	2
15	if in if with integer	24	2	2
16	arithmetic with if	11	2	2
17	if in if with character	27	2	4
18	while loop	19	1	8
19	structure data setting	18	1	7
20	output format specifier	11	1	4
21	memory address with pointer for character array	10	4	11
22	memory address with pointer for various array	24	1	13
23	four arithmetic operations	11	1	4
24	greatest common divisor (GCD)	21	1	4
25	inner product operation	16	1	1
26	linear search	23	1	5
27	binary search	29	1	17
28	linked list	67	1	7
29	tree traversal algorithms	81	3	63
30	stack data structure	47	1	8
31	reverse polish notation	67	2	12
32	queue data structure	55	1	4
33	insertion sort algorithm	25	1	30
34	selection sort algorithm	28	1	18
35	bubble sort algorithm	27	1	37
36	shell sort algorithm	24	1	23
37	merge sort algorithm	34	1	16
38	quick sort algorithm	38	1	56
39	heap sort algorithm	44	1	42
40	pointer	12	1	6
41	data size of pointer	20	1	9
42	list structure	47	1	12
maximum		81	4	63
minimum		6	1	1
average		24.50	1.26	13.95
total		1029	53	586

```
#include <stdio.h>
main()
{
    char vch, *pch;
    int vin, *pin;
    double vdo, *pdo;
    printf("sizeof(vch) = %d\n", sizeof(vch));
    printf("sizeof(vin) = %d\n", sizeof(vin));
    printf("sizeof(vdo) = %d\n", sizeof(vdo));

    printf("sizeof(pch) = %d\n", sizeof(pch));
    printf("sizeof(pin) = %d\n", sizeof(pin));
    printf("sizeof(pdo) = %d\n", sizeof(pdo));

    printf("sizeof(*pch) = %d\n", sizeof(*pch));
    printf("sizeof(*pin) = %d\n", sizeof(*pin));
    printf("sizeof(*pdo) = %d\n", sizeof(*pdo));
}
```

Figure 5: Source code for VTP instance at *ID=41*.

```
(64-bit size system)
What is the output of this program?
Data size of normal variables
sizeof(vch) = _1_
sizeof(vin) = _2_
sizeof(vdo) = _3_

Data size of pointer
sizeof(pch) = _4_
sizeof(pin) = _5_
sizeof(pdo) = _6_

Data size of pointer reference
sizeof(*pch) = _7_
sizeof(*pin) = _8_
sizeof(*pdo) = _9_
```

Figure 6: Questions for VTP instance at *ID=41*.

4.3 Hints in Answer Interface

On *hints* for memory addresses, Figures 7-9 show the source code, the hints, and the questions for the VTP instance at *ID=22*. The hints describe the byte size of each data type with the word size that is necessary to calculate the increase of the memory address, and the memory addresses of the first two elements in the array variables, so that students can answer the question on them.

```
#include <stdio.h>
int main(void){
    int vch, vsh, vin, vfl, vdo;
    vch = sizeof(char);
    vsh = sizeof(short);
    vin = sizeof(int);
    vfl = sizeof(float);
    vdo = sizeof(double);

    printf("%d %d %d %d %d \n", vch, vsh, vin,
        vfl, vdo);//hintA

    char arr1[] = {'o','k','a','y','a','m','a'};
    int arr2[5] = {1,2,3,4,5};
    float arr3[5] = {1.1,2.2,3.3,4.4,5.5};
    double arr4[5] = {1.0,2.2,3.5,22.4,10.0};

    printf("arr1[4] is %c \n", arr1[4]);
    printf("arr2[3] is %d \n", arr2[3]);
    printf("arr3[2] is %f \n", arr3[2]);
    printf("arr4[1] is %f \n", arr4[1]);

    printf("The address of arr1[2] is %p \n", &
        arr1[2]);//hintB
    printf("The address of arr2[2] is %p \n", &
        arr2[2]);//hintC
    printf("The address of arr3[3] is %p \n", &
        arr3[3]);//hintD
    printf("The address of arr4[2] is %p \n", &
        arr4[2]);//hintE
    return 0;
}
```

Figure 7: Source code for VTP instance at *ID=22*.

```
hintA: Data types and sizes(32-bit system)
char: 1 byte
short: 2 bytes
int 4: bytes
float: 4 bytes
double: 8 bytes

hintB: The address of arr1[0] is 0028FE30
       The address of arr1[1] is 0028FE31

hintC: The address of arr2[0] is 0028FE20
       The address of arr2[1] is 0028FE24

hintD: The address of arr3[1] is 0028FEF4
       The address of arr3[2] is 0028FEF8
hexadecimal: the symbols "0"~"9" to represent
    values zero to nine, and "A"~"F" to represent
     values ten to fifteen.

hintE: The address of arr4[0] is 0028FE10
       The address of arr4[1] is 0028FE18
```

Figure 8: Hints for VTP instance at *ID=22*.

```
What is the output of this program?
_1_ _2_ _3_ _4_ _5_
arr1[4] is _6_
arr2[3] is _7_
arr3[2] is _8_
arr4[1] is _9_
The address of arr1[2] is _10_
The address of arr2[2] is _11_
The address of arr3[3] is _12_
The address of arr4[2] is _13_
```

Figure 9: Questions for VTP instance at *ID=22*.

4.4 Multiple Choice Questions

On *multiple choice questions* for linked lists, Figures 10 and 11 show the source code and the questions for the VTP instance at *ID=28*. The multiple choice questions of the correct answers are used at Q1 and Q3 as the hints to understand the pointer and structure in the linked list.

```
#include <stdio.h>
#include <errno.h>
#include <string.h>
#include <time.h>
#include <stdlib.h>
struct cell{
    struct cell *next;
    int data;
};
typedef struct cell cell_t;
cell_t *list_alloc(int data){
    cell_t *new = NULL;
    new = (cell_t *)malloc(sizeof(cell_t));
    if(new == NULL){
        fprintf(stderr, "ERROR: list_alloc(): %s\
           n", strerror(errno));
        return(NULL);
    }
    new->next = NULL;
    new->data = data;
    return(new);
}
int list_add(cell_t *header, int data){
    cell_t *next = NULL;
    cell_t *prev = header;
    next = list_alloc(data);
    if(next == NULL) return(-1);
    while(prev->next != NULL){
        prev = prev->next;
    }
    prev->next = next;
    return 0;
}
void list_free(cell_t *header){
    cell_t *temp = header;
    cell_t *swap = NULL;
    while(temp != NULL){
        swap = temp->next;
        free(temp);
        temp = swap;
    }
}
static void list_print(cell_t *header){
    cell_t *p = header;
    printf("list{ ");
    while(p != NULL){
        printf("%d ", p->data);
        p = p->next;
    }
    printf("}\n");
}
int main(void){
    int cnt = 0;
    cell_t *header = list_alloc(0);
    if(header == NULL) return -1;
    for(cnt = 1; cnt < 5; cnt++){
        list_add(header, cnt*7%10);
    }
    list_print(header);
    list_free(header);
    return 0;
}
```

Figure 10: Source code for VTP instance at *ID=28*.

```
Q1: Select one word for each blank from a list of
     choices.
What is data type of *header? _1_
What is data type of header? _2_
(A) pointer
(B) data
(C) cell
(D) cell_t

Q2: What is "data" of the cell pointed by "header
    " in the code?  _3_
What is "next" of the cell pointed by "header" in
     the code?  _4_

Q3: Select one word for each blank from the list
    of choices given below from list_free method.
What is the name of cell that will be freed this
    time? _5_
What is the name of cell that will be freed next
    time? _6_
(A) swap
(B) temp

Q4: In list_add method, what is the name of new
    cell? _7_

Q5: What is the output of this programming?
list { _8_ _9_ _10_ _11_ _12_ }
```

Figure 11: Questions for VTP instance at *ID=28*.

4.5 Links to Websites in Answer Interface

On *links to reference websites* for algorithms, the links to the websites that explain *reverse polish notation* in [9] and *quick sort algorithm* in [10] are included in the answer interface. By clicking these links, the students can easily access to the necessary information to solve them.

5 Evaluations

In this section, we evaluate the generated 42 VTP instances with 586 answer forms by applying them to 49 undergraduate students in Myanmar (21 students), Japan (18 students), and China (10 students) who are currently studying or have studied *C programming* and *fundamental data structures & algorithms*.

5.1 Solution Results for Individual VTP Instances

First, the solution results for the 42 individual VTP instances by the 49 students are analyzed. Figure 12 shows the total number of answer submission times and the average correct answer rate (%) for each VTP instance submitted by the students. As the summary, Table 2 shows the maximum, minimum, average, and standard deviation (SD) of the total number of submissions and the average correct answer rate among the 42 VTP instances. The average correct rate 94.29% and the average number of submissions 139.17 by all the students (2.84 by one student) for each instance suggest that the generated VTP instances are of moderate difficulty for novices to study *C programming*.

Table 2: Summary of solution results for individual VTP instances.

\	total# of submissions	average correct rate
maximum	628	100 %
minimum	55	82.32 %
average	139.17	94.29%
SD	106.14	4.84%

5.1.1 VTP Instances with Low Results

The VTP instances for *pointer* or *fundamental data structures & algorithms* are generally difficult for the students. For the eight instances at *ID=21*, *24*, *28*, *29*, *30*, *32*, *36*, and *37*, which are on *pointer*, *stack*, *linked list*, and *sorting*, the average correct answer rate is under 90%.

For the VTP instance *ID=41 linked list*, the correct answer rate 82.32% is lowest and the number of answer submissions 628 is highest among the 42 instances. This topic is still hard for the students, regardless of the presented devices in Section 4. However, it is observed that the students tried to solve this instance by referencing to the devices.

For the VTP instance for *merge sort* at *ID=37*, both the correct rate 85.87% and the number of submissions 101 are relatively low. Many students gave up solving it correctly at early time, which suggests their insufficiency in understanding this algorithm. Like for *quick sort*, the device of adding the link to the reference website in the answer interface will be necessary for this instance.

5.1.2 Distribution of Correct Answer Rates

Figure 13 shows the distribution of the correct answer rates among the 42 VTP instances. 34 instances achieved more than the 90% correct rate and seven did 100%, which indicates that these instances are moderate for novice students. However, the rates of eight instances were less than 90%, which need to be improved.

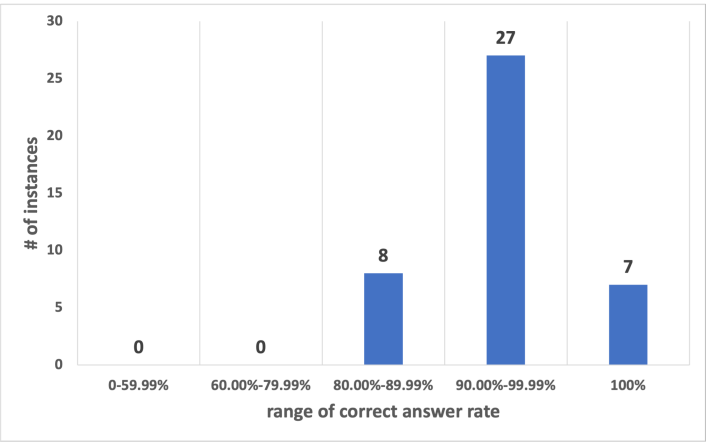

Figure 13: Distribution of correct answer rates among instances.

5.1.3 Distribution of Submission Times

Figure 14 shows the distribution of the number of answer submission times among the VTP instances. 21 instances were solved within 100 submissions. 30 instances were within 150 submissions by all the students or 3.57 submissions by one student. However, for the three instances at *ID=22*, *28*, and *29*, the answers were submitted more than 300 times, which will need to be improved.

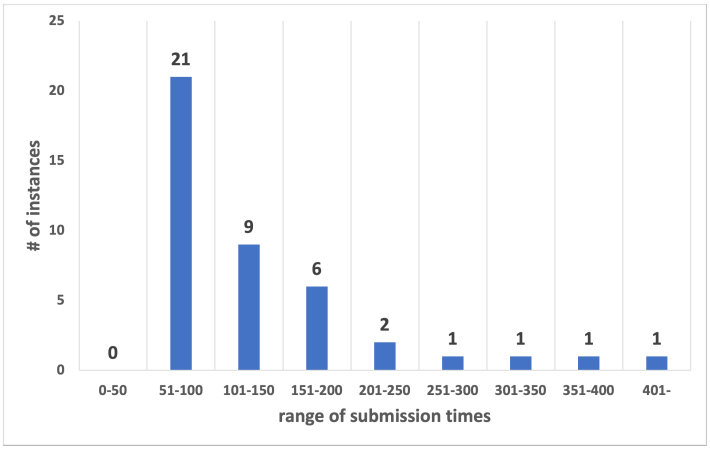

Figure 14: Distribution of submission times among instances.

5.2 Solution Results for Individual Students

Next, the solution results for the individual students to the 42 VTP instances are analyzed. Figure 15 shows the total number of answer

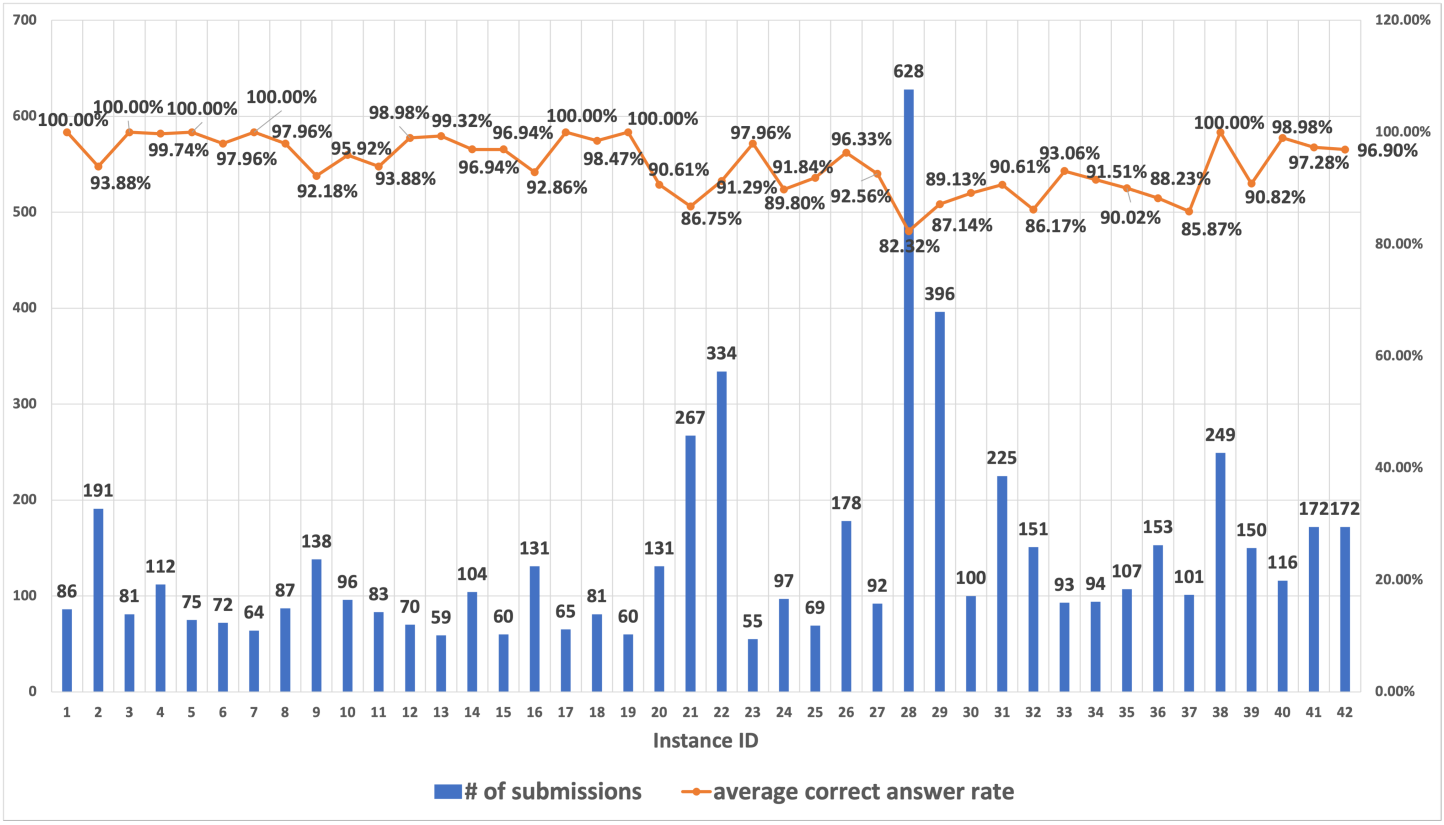

Figure 12: Solution results for individual VTP instances.

submissions and the average of the correct answer rates among all the instances for each student. Table 3 shows the summary results for each student. The results suggest that most of the students well solved the VTP instances.

Table 3: Summary of solution results for individual students.

	total # of submissions	correct answer rate
maximum	394	100%
minimum	38	58.75%
average	119.29	94.29%
SD	89.61	8.25%

5.2.1 *Students with Low Results*

For 11 students at *ID=8*, *10*, *11*, *12*, *13*, *15*, *18*, *19*, *33*, *46*, and *49*, the average correct answer rate among the 42 VTP instances is smaller than 90%.

For the student at *ID=18*, both the correct answer rate 58.75% and the number of answer submissions 38 are lowest among the students. Even, this student did not try to solve some instances. The teacher should find the reason and help this student studying *C programming*. The five students at *ID=10*, *12*, *33*, *46*, and *49* also have the similar proclivities.

For the two students at *ID=8* and *15*, the number of submissions is relatively high. Thus, they made sufficient efforts to solve the VTP instances. It can be expected that the proper guidance by the teacher can quickly improve their performances.

5.2.2 *Distribution of Correct Answer Rates*

Figure 16 shows the distribution of the correct answer rates among the 49 students. 38 students achieved over 90% correct answer rates. Eight students at *ID=22*, *24*, *26*, *27*, *28*, *32*, *39*, and *44* achieved 100%.

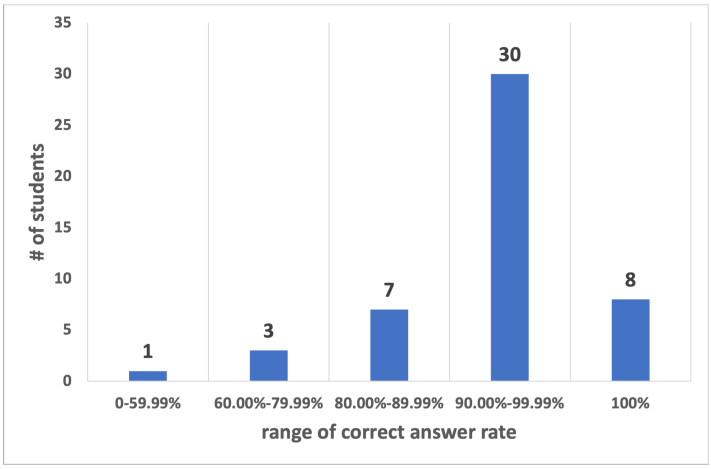

Figure 16: Correct answer rate distribution of students.

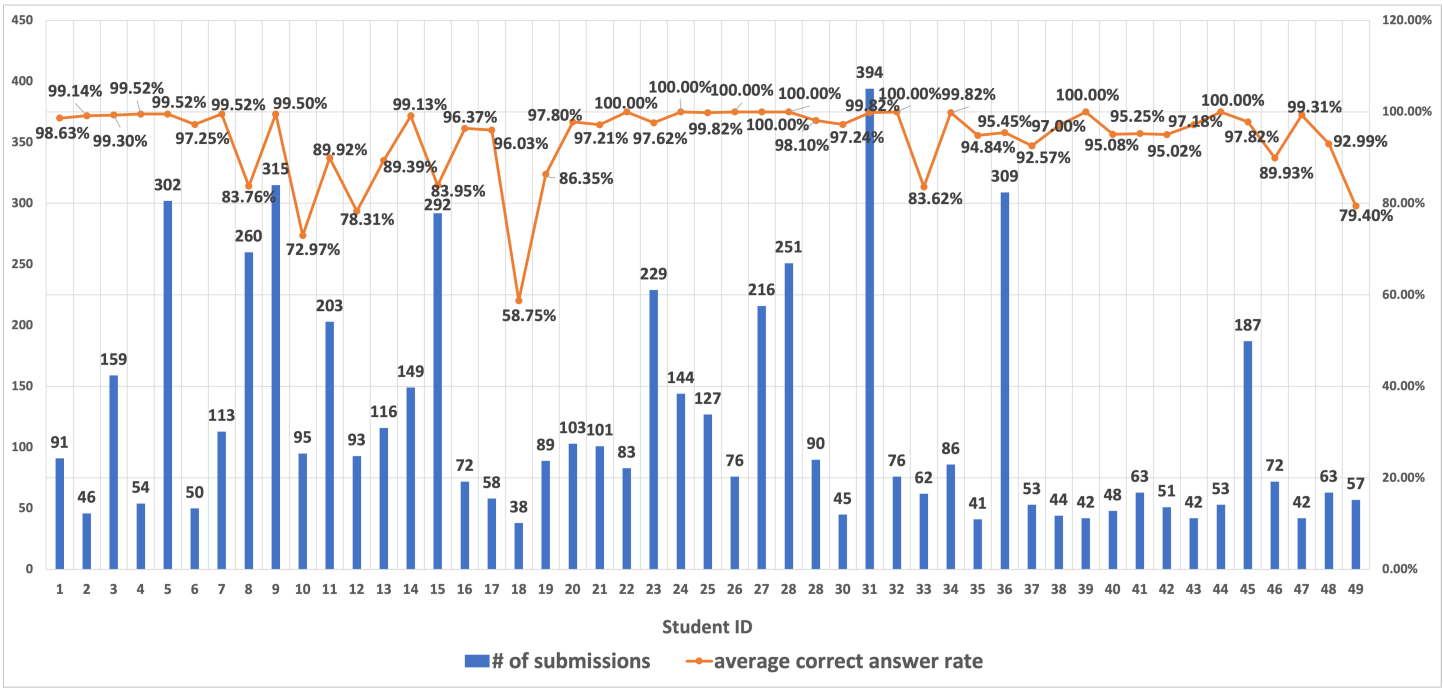

Figure 15: Results for each student.

5.2.3 *Distribution of Submission Times*

Figure 17 shows the distribution of the number of answer submission times among the students. For 30 students, the total number of submissions was smaller than 100, where the excellent student at *ID=39* correctly solved any instance except one instance by submitting the answers only one time. For the four students at *ID=5*, *9*, *31*, and *36*, the total number of submissions exceeded 300, and the answer rate is larger than 90%. It was found that these students did not give up solving them.

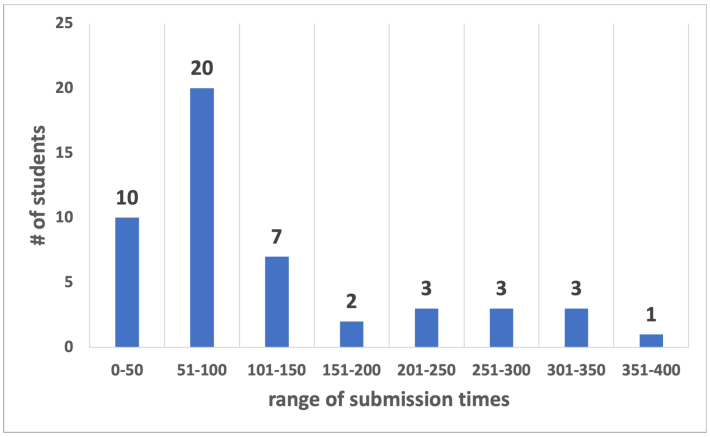

Figure 17: Submission times distribution among students.

5.3 *Evaluations of Devices for Hard Instances*

To evaluate the effectiveness of the presented devices in Section 4, we compare the solution results *with* and *without* using them for the four hard VTP instances at *ID=22*, *28*, *31*, and *38*.

5.3.1 *Comparison Results*

In our preliminary application, we applied the four instances to 21 students among 49 *without* using the proposed devices. Then, to avoid the solution improvements by solving the same instances twice by the same students, the results of the remaining 28 students are used as the results *with* them.

Table 4 compares the average number of answer submission times and the average correct answer rate for the four instances by the students between the *with* group and the *without* group. In the *without* group, the average correct answer rate for any instance was smaller than 70%, which suggests that many students suffered from solving them. Then, in the *with* group, it was improved to higher than 80%, which can be considered as the minimum rate that every student should exceed. The average number of submissions was increased except for *ID=31*, because the students were motivated to continue solving them due to the devices.

Table 4: Comparisons of solution results for hard instances.

ID	topic	average # of submissions		average correct answer rate	
		with	*without*	*with*	*without*
22	memory address with pointer for various array	7.95	7.17	91.29%	54.40%
28	linked list	14.95	1.92	82.32%	56.00%
31	reverse polish notation	5.36	6.88	90.61%	52.00%
38	quick sort algorithm	5.93	1.71	100.00%	68.79%
average		8.55	4.42	91.06%	57.80%

5.3.2 t-test *Analysis*

t-test is applied to the solution results to check the significant difference between the average correct answer rates of the two groups. The *p-value* is $1.179E-16$ and is smaller than 0.05, which means that there is the significant difference between them. Thus, it is confirmed that the proposed devices for the hard VTP instances are effective in improving the solution performances of novice students.

5.4 *Discussions*

At solving the VTP instances, some students may reach the correct answers without understanding them well. They may randomly submit possible answers or copy the answers of other students. Fortunately, the answer interface can record any submitted answer of a student. By analyzing the records, it may be possible to find the undesirable behaviors of students, which will be in our next study.

6 Conclusion

This paper presented the *value trace problem (VTP)* for independent *code reading study* of *C programming*. 42 VTP instances were generated using simple C source codes for *basic grammar concepts* and *fundamental data structures & algorithms* in textbooks or websites. Besides, the devices of tips, hints, multiple choice questions, and links to reference websites were presented for hard instances on *pointer* and *algorithms* to improve the solution performances.

For evaluations, the generated 42 VTP instances were assigned to 49 undergraduate students in Japan, China, and Myanmar to be solved at home using the answer interface that can run offline. The average correct answer rate among all the VTP instances reached 94.29%, where by our proposed devices, the average rate for the four hard instances was improved by 33.26% from 57.80% before improvements. Thus, the effectiveness of our proposal was confirmed in motivating *self-study* of *C programming* to novice students at home.

A limitation of *VTP* is the manual selection of variables or output messages in the source code for questions, in addition to the manual collection of C source codes. An algorithm should be investigated to automatically generate a new VTP instance from a given source code. Another limitation is the impossibility of practicing source code writing by students.

As future works, we will study the automatic VTP generation algorithm, make new VTP instances using C source codes for other grammar concepts and algorithms, such as file I/O, error handing, recursions, and graph algorithms, and apply them to students in *C programming* courses. Then, more comprehensively and formally, we will study the methodologies of organizing and conducting pedagogical experiments to verify the effectiveness of the proposal in increasing motivations of novice students to study *C programming*, and the methods of processing the results of pedagogical experiments.

References

[1] Sio-Iong Ao, IAENG Transactions on Engineering Sciences - Special Issue for the International Association of Engineers Conferences 2016 Volume II, 2018.

[2] N. Funabiki, K.K. Zaw, W.-C. Kao, "A proposal of value trace problem for algorithm code reading in Java programming learning assistant system," Information Engineering Express, 1(3), 2015, doi:10.52731/iee.v1.i3.39.

[3] K.K. Zaw, W. Zaw, N. Funabiki, W.C. Kao, "An informative test code approach in code writing problem for three object-oriented programming concepts in java programming learning assistant system," IAENG International Journal of Computer Science, **46**(3), 2019.

[4] N. Funabiki, H. Masaoka, N. Ishihara, I.W. Lai, W.C. Kao, "Offline answering function for fill-in-blank problems in Java Programming Learning Assistant System," in 2016 IEEE International Conference on Consumer Electronics-Taiwan, ICCE-TW 2016, 2016, doi:10.1109/ICCE-TW.2016.7521045.

[5] Pointer and gcc, https://ubuntuforums.org/archive/index.php/t-858030.html.

[6] FCP 2 If else.ppt, https://dokumen.tips/documents/fcp2ifelseppt.html.

[7] Loop and break, https://www.loopandbreak.com.

[8] T. Hikita, Algorithms by C, Science Pub., 1995.

[9] Reverse polish notation, http://www.cs.man.ac.uk/ pjj/cs212/fix.html.

[10] Quick Sort, https://www.programiz.com/dsa/quick-sort.

[11] M. G`ómez-Albarr`án, The teaching and learning of programming: A survey of supporting software tools, Computer Journal, **48**(2), 2005, doi:10.1093/comjnl/bxh080.

[12] P. Brusilovsky, S. Sosnovsky, "Individualized Exercises for Self-Assessment of Programming Knowledge: An Evaluation of QuizPACK," ACM Journal on Educational Resources in Computing, **5**(3), 6, 2005, doi:10.1145/1163405.1163411.

[13] M. Kordaki, "A drawing and multi-representational computer environment for beginners' learning of programming using C: Design and pilot formative evaluation," Computers and Education, **54**(1), 69–87, 2010, doi:10.1016/j.compedu.2009.07.012.

[14] M.J. Lee, A.J. Ko, "Personifying programming tool feedback improves novice programmers' learning," in ICER'11 - Proceedings of the ACM SIGCSE 2011 International Computing Education Research Workshop, 2011, doi:10.1145/2016911.2016934.

[15] F.W.B. Li, C. Watson, "Game-based concept visualization for learning programming," in MM'11 - Proceedings of the 2011 ACM Multimedia Conference and Co-Located Workshops - ACM International Workshop on Multimedia Technologies for Distance Learning, MTDL'11, 2011, doi:10.1145/2072598.2072607.

[16] W.Y. Hwang, R. Shadiev, C.Y. Wang, Z.H. Huang, "A pilot study of cooperative programming learning behavior and its relationship with students' learning performance," Computers and Education, **58**(4), 2012, doi:10.1016/j.compedu.2011.12.009.

[17] D.M. Souza, K.R. Felizardo, E.F. Barbosa, "A systematic literature review of assessment tools for programming assignments," in Proceedings - 2016 IEEE 29th Conference on Software Engineering Education and Training, CSEEandT 2016, Institute of Electrical and Electronics Engineers Inc.: 147–156, 2016, doi:10.1109/CSEET.2016.48.

[18] S. Parihar, R. Das, Z. Dadachanji, A. Karkare, P.K. Singh, A. Bhattacharya, "Automatic grading and feedback using program repair for introductory programming courses," in Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE, Association for Computing Machinery: 92–97, 2017, doi:10.1145/3059009.3059026.

[19] X. Fu, A. Shimada, H. Ogata, Y. Taniguchi, D. Suehiro, "Real-time learning analytics for C programming language courses," in ACM International Conference Proceeding Series, Association for Computing Machinery: 280–288, 2017, doi:10.1145/3027385.3027407.

[20] T. Busjahn, C. Schulte, "The use of code reading in teaching programming," in ACM International Conference Proceeding Series, 3–11, 2013, doi:10.1145/2526968.2526969.

[21] T. Vandegrift, "Reading before writing: can students read and understand code and documentation?," SIGCSE '05 Cojoined Meeting, 2005.

[22] D. Kwon, I. Yoon, and W. Lee, "D.Y. Kwon, I.K. Yoon, W.G. Lee, "Design of programming learning process using hybrid programming environment for computing education," KSII Transactions on Internet and Information Systems, **5**(10), 1799–1813, 2011, doi:10.3837/tiis.2011.10.007.

[23] T. Busjahn, R. Bednarik, C. Schulte, "What influences dwell time during source code reading? Analysis of element type and frequency as factors," in Eye Tracking Research and Applications Symposium (ETRA), 2014, doi:10.1145/2578153.2578211.