Chen, W. et al. (Eds.) (2016). Proceedings of the 24th International Conference on Computers in Education. India: Asia-Pacific Society for Computers in Education

Learning Support System for Visualizing Memory Image and Target Domain World, and Classroom Practice for Understanding Pointers

Koichi YAMASHITA[a*], Ryota FUJIOKA[b], Satoru KOGURE[c], Yasuhiro NOGUCHI[c], Tatsuhiro KONISHI[c] & Yukihiro ITOH[d]
[a]*Faculty of Business Administration, Tokoha University, Japan*
[b]*Graduate School of Informatics, Shizuoka University, Japan*
[c]*Faculty of Informatics, Shizuoka University, Japan*
[d]*Shizuoka University, Japan*
*yamasita@hm.tokoha-u.ac.jp

Abstract: In this paper, we describe a learning support system that can visualize program behavior in memory image and status of the target domain world, and classroom practice that introduces the system to a real class. Several learning support systems have been developed for supporting novice learners in understanding program codes. However, visualization in existing systems often conceals the concrete value of variables such as pointers; the way in which each visualized object is located on the memory is not made explicit. In order to address this issue, we developed a learning support system that visualizes simultaneously and synchronously the memory image that is the field that presents the concrete value of variables, and the target domain world that is the field that presents logically the data structures processed by the program. Moreover, in order to allow teachers to set their instruction content based on the growing variety of learner background knowledge, our system visualizes the status of the target domain world according to the visualization policy defined by the teacher. We conducted classroom practice for understanding pointers in connection with a memory model, thus introducing our system to a real class. In this practice, we included a learning step for the students to observe the memory image of variables and compare it with the status of the target domain world, which had not been included in our preceding practice. Analysis of answered scores in a questionnaire conducted after the practice suggests that our system contributed to reducing irregularities in the participants' understanding, and that appropriate learning support reached every participant of our practice using our system.

Keywords: Education for programming, Learning support system, Domain world model, Classroom practice.

1. Introduction

As information technology has pervaded our society in recent years, improving the productivity of program codes has been expected increasingly. Many institutes of higher education have organized programming classes for various students. Thus far, several learning support systems have been developed to support novice learners in understanding program codes (Pears et al., 2007; Malmi et al., 2004; Moreno, Myller, Sutinen, & Ben-Ari, 2004; Neve, Hunter, Livingstone, & Orwell, 2012; Yamashita et al., 2016). These systems visualize the data structures processed by the target programs in certain ways, and support learners in understanding the programs and algorithms by making their behavior visible. When understanding algorithms and programs, having a clear image about their behavior is important. However, novice learners often find it difficult to obtain such image correctly. Hence, introducing these systems to classes is expected to allow learners to cultivate a better understanding of program (Naps et al., 2002).

The visualization of data structures is also used in classroom lectures on algorithms. Teachers often explain the behavior of an algorithm by drawing data structures with a concept that is specific to the target algorithm, such as trees, lists, and stacks. However, such visualized structures often cannot be

immediately expressed by some lexical items and syntactic fragments provided by the programming language. Hence, there are some discrepancies between the visualizations and program code. An expert programmer might have an image of the status of a target domain world according to the memory image of the variables provided by a debugger or tracer, and might grasp changes in the world status based on the program code. We consider that the failure of many of novice programmers to understand programs is caused by their poor ability to perceive the relationships among the program-code, memory image, and status of the target domain world.

Based on this consideration, we developed a learning support system called TEDViT (Teacher Explaining Design Visualization Tool) that visualizes the data structures processed by target programs in two ways: memory image of variables, and status of the target domain world (Kogure et al., 2014). Although a typical debugger or tracer only provides the values of the variables on memory, and typical existing systems only provide data structures with a status of the target world, TEDViT allows learners to observe and compare both visualizations. In addition, TEDViT visualizes the target domain world according to the visualization policy defined by a teacher. This function allows teachers to set the content or intent of instructions, such as the point where the learners should focus on the algorithm or program, or the abstraction or generalization degree of instruction, based on the learners. Almost all of the existing systems disallow these variations of teacher intent, and visualize the target domain world in a constant visualization policy.

We conducted some classroom practices with TEDViT for various learners with different background knowledge. In (Yamashita et al., 2015), we described the classroom practice incorporated into the lecture course "Algorithm" for university students in a business administration major. In there, it was suggested that conducting discovery learning about the properties of algorithm using TEDViT contributed the students' understanding of algorithm. However, we did not include a learning step for the students to observe the memory image of variables and compare it with the status of the target domain world because the main objective of the practice was to understand the behavior of the target algorithms and their differences.

In this paper, we describe a classroom practice for software engineers aimed at allowing them to cultivate a better understanding of pointers in connection with memory models, following the description of our system TEDViT. We also describe the questionnaire conducted after the practice and an analysis of the answers. The evaluation results suggest that TEDViT would have a certain degree of effectiveness in learning and teaching programming and in programming education.

2. TEDViT: Teacher Explaining Design Visualization Tool

TEDViT is the system we developed in our previous work (Kogure et al., 2014) and has two characteristic features that are different from existing systems. First, TEDViT can visualize the data structures processed by target programs in two ways: memory image of variables, and status of the target domain world. Second, TEDViT can visualize the target domain world according to visualization policy defined by a teacher. In this section, we describe these two features of TEDViT.

2.1 Visualizing Memory Image of Variables and Status of Target Domain World

Several learning support systems have been developed for supporting novice learners in understanding program codes. These systems visualize the data structures processed by target programs in certain ways, and support learners in understanding the programs and algorithms by making their behavior visible. For example, iList (Fossati, Eugenio, Brown, & Ohlsson, 2008) visualizes linked lists with graphical objects, including rectangle boxes for list nodes and arrowed connectors for pointer values. Its visualization consists of logical data structures based on particular concepts, many of which are specific to the target algorithm.

However, such visualized structures often cannot be immediately expressed by some lexical items and syntactic fragments provided by the programming language. Moreover, the concrete values of variables are often concealed, and hence the way in which each visualized object is located on memory is not made explicit. Therefore, novice learners often fail to grasp the relationship between behavior and program code, and reach a learning impasse.

To address this issue, TEDViT visualizes the data structures processed by a target program in two ways, memory image of variables and status of target domain world, and provides a learning environment where learners can observe and compare both visualizations. Figure 1 shows a learning environment generated by TEDViT. The environment consists of three fields: the data structures processed by the program in (A) are visualized in the two fields in (B) and (C). TEDViT reproduces a series of memory images of variables in (B) for each step of the program's execution, and a series of statuses of the target domain world in (C) that visualizes logical data structures.

When a learner clicks the "Next" or "Prev" button, the highlight in (A) moves to the next or previous statement in the program code; the memory image in (B) is updated according to the values of the variables after executing the highlighted statement; and the corresponding status of the target domain world is visualized in (C). TEDViT simulates statement execution step by step so that the learner can understand the program's behavior by observing the changes of the target domain world in (C). Simultaneously, the learner can understand the concrete memory image in each execution step and the concrete expression of the data structures by observing and comparing the world in (C) with the memory image in (B).

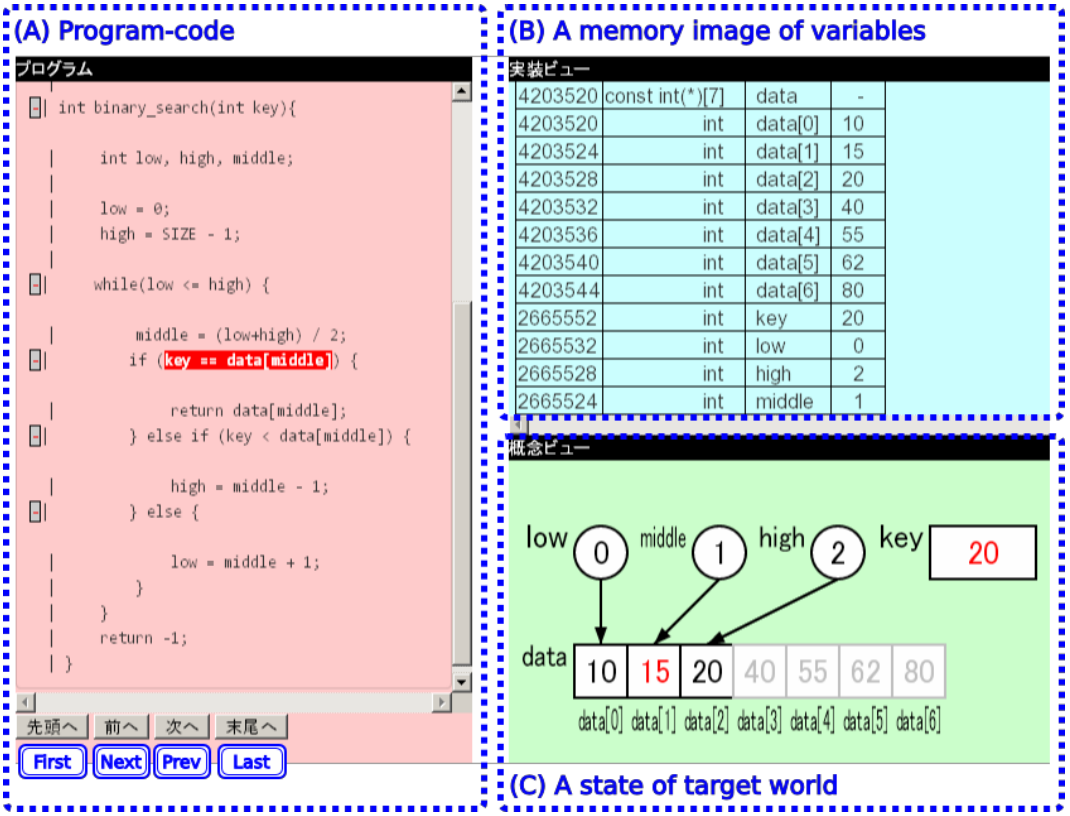

Figure 1. Overview of learning environment generated by TEDViT.

2.2 Visualizing Status of Target Domain World Based on Teacher's Intention

For example, a teacher might draw an object in a horizontal layout when the instruction target is to sort an array, as shown in Figure 2, whereas the teacher might draw the array in a vertical layout for a stack. Changes in visualization policy such as this are derived by fitting the instruction content to the learners' background knowledge. For example, if the learners sufficiently understand a stack, drawing either object in a horizontal or vertical layout would be acceptable to the learners. Similarly, the teacher would not need to draw the temporary variable in a task that swaps the values of two variables for non-novice learners.

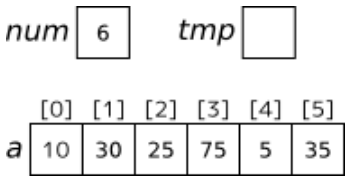

Figure 2. Example of a status of target world.

A typical method for providing visualized data structures to learners is to show slides and/or movies made with presentation and video editing software. Nevertheless, these materials cannot be used for certain learning activities, such as learners observing program behavior where input data are changed individually, because the input data are fixed. Other method to do with allowing learners to change input data is to provide target program to learners, including graphic drawing functions. However, this is also not realistic because it might burden teachers with troublesome coding, such as creating, updating, and deleting drawing objects with name resolution that involves namespaces, scopes, and so on.

To resolve this problem, we implemented a function for teachers to define the policy for drawing a status of the target domain world according to their own intent. The teachers can create or edit a configuration file independently from the target program file. TEDViT interprets such visualization policy by scanning the configuration file and visualizes the target domain world according to it. The learners can then observe the program behavior in the target world visualized in accordance with the teacher's intent. The relationship among teacher, learner, and TEDViT is shown in Figure 3.

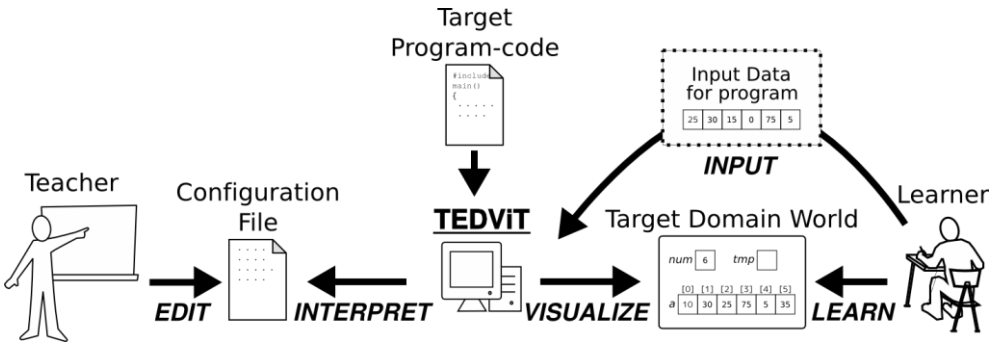

Figure 3. Relationship among teacher, learner, and TEDViT.

2.3 Defining Visualization Policy for Target Domain World

A configuration file that defines visualization policy consists of a set of drawing rules, each of which consists of the following information:

- Condition to actuate the drawing operation
- Operation to edit the target drawing object
- Identifier of the target drawing object
- Type of target drawing object
- Attributes for the drawing operation

We selected available object and attribute types by investigating actual teaching materials for algorithm and programming education in the faculties to which the authors belong. Table 1 provides a list of the available types, operations, and attributes. Circle, square, and rectangle objects are mainly used to express directly the value of the variable in the target program. The table object mainly expresses an array. Connector and line objects express a relationship between the two objects, and label and balloon objects are used to describe program behavior or object role in natural language.

Table 1: Types of object, drawing operation, and attribute for configuration.

Objects	Operations	Attributes
Circle	Create	Corresponding variable*
Square	Delete	Main object ID**
Rectangle	Update	Position
Table		Width
Connector		Height
Line		Color
Label		Line weight
Balloon		Line style

* Only for circle, square, rectangle and table object.
** Only for connector, line and balloon object.

Teachers can describe a condition for actuating the drawing operation by referencing the statement number (statement ID) or variables in the target program, such as "when a certain statement is executed," or "when the value of a certain variable satisfies the condition." A condition can be expressed with six types of comparison operators, ==, !=, >=, <=, <, and >, and three types of operands, immediate number, variable in target program, and statement ID.

Some examples of drawing rules that include such information are described in Figure 4 in CSV format. The description in the first line is a rule that when the statement with ID "3" in the target program is executed, TEDViT draws a circle object and assigns the object ID "OBJ1" to it. The corresponding variable is described as "*low*," and hence the value of the variable *low* is drawn inside OBJ1. OBJ1 is drawn at position ("X1," "Y1") with a line, background, and inner character colors of "black," "white," and "black," respectively, according to the description in the rule. The rule in the second line means that when the statement with ID "5" is executed, the line color of the already visualized object "OBJ1" is updated to "red." The last line is the rule that when variable "*j*" has a value that is greater than "*i*" in the target program, the inner character color of "OBJ1" is updated to "blue."

```
state==3,create,OBJ1,circle,low,X1,Y1,black,white,black
state==5,update,OBJ1,,,,,red,,
j > i,update,OBJ1,,,,,,,blue
```

Figure 4. Examples of drawing rule descriptions in configuration file.

2.4 Preliminary Experiment

We conducted a small preliminary experiment in order to evaluate the difficulty of preparing teaching material by the rule definitions in TEDViT. The subjects in the experiment were one master's student and two undergraduate students with teaching assistant abilities. We measured the time required for the subjects to prepare teaching materials by the rule definitions in following manner: first, we provided each subject with sufficient tutorials to make teaching materials by the rule definitions in TEDViT. Next, we presented the subjects with the program code for selection sort and requested them to describe the rule set required to visualize the program behavior in the target domain world. In our request, we conveyed that each subject has to implement the visualization policy pre-established by us. After completing the rule definition, we requested the subjects to create slide materials using PowerPoint with the same content and behavior as the materials for TEDViT. Table 2 provides the times required for each subject to complete each procedure.

Table 2: Times required to complete each procedure.

	Subject A	Subject B	Subject C
Tutorial	56 min.	43 min.	52 min.
Rule definitions	32 min.	30 min.	33 min.
Slide creations	23 min.	33 min.	29 min.

Although the number of subjects was small, the result of this experiment suggests that teaching materials for TEDViT could be prepared in practical time with some experience in rule definitions. The times required to complete rule definitions are approximately the same as slide creations. However, the teaching material for TEDViT would be more useful because TEDViT can reproduce the program behavior without rule modifications, even if the target data processed by the program change.

3. Classroom Practice for Understanding Pointers

Thus far, we have conducted several educational practices with TEDViT incorporated in real classes. In (Yamashita et al., 2015), we presented algorithm classroom practices for students in a business administration major. The practices were based on discovery learning on the properties of algorithms, such as the number of comparisons and swaps. TEDViT supported the students to construct their discovery learning cycles by visualizing some suggestive objects in the target domain world in order to provide learning guidance, and by allowing the students to input data in the target program based on their hypotheses. The evaluation results based on the score improvements between pre and post-tests suggest that the practices contributed, to a certain degree, to the students' understanding of algorithms.

Nevertheless, the preceding practices did not include learning activities for observing memory images in the learning scenario because the goal was to understand algorithm behaviors and the differences among the target algorithms based on their properties. In this section, we describe the classroom practice that used TEDViT for understanding pointers. The practice was conducted along with a scenario that included activities for observing and comparing memory images and the target domain world.

3.1 Classroom Practice Overview

Hamamatsu Embedded Programming Technology Consortium (HEPT) is a collaborative consortium with industries and the university to which some of the authors belong; HEPT is also training system engineers. HEPT offers a training course for software design and development engineering, including lectures on programming in C. Our classroom practice was incorporated into a session of these lectures. The practice participants were 15 software engineers responsible for software development, but without much experience in C programming, as listed in Tables 3 and 4.

Table 3: Number of participants with different years of C programming experience.

Years of experience	Less than 1 year	1-3 years	3-5 years	5-7 years
Number of participants	5	7	1	2

Table 4: Number of participants with experience in each language (multiple replies allowed).

Programming Language	C	C++	C#	Visual Basic	Java	Perl
Number of participants	13	6	3	4	4	1

The goal of the session is to understand how to design and implement a function that can take arbitrary data types using function and generic pointers. In order to achieve this goal, the learners need to understand "relationships among types, variables, and pointers," "pointers in function arguments," "pointers as return values," "pointer operations," and "relationships between pointers and arrays" in connection with a memory model. However, it is difficult for novice learners to understand pointers with a memory model. Hence, we introduced TEDViT into the session class to support learners in their understanding.

In the practiced session, the teacher taught the following learning sections in order:

1. Variables and pointers
   a. Pointer operators and memory model
2. Functions and pointers
   a. Calls by value
   b. Calls by pointer

c. Return values with pointers

3. Arrays and pointers
   a. Relationship between array names and pointers
   b. Relationship between arithmetic operations on pointers and array index operations
   c. Memory model of multidimensional arrays
   d. Handling an array to reverse the order
4. Generic pointers
   a. Operations on generic pointers
   b. Sample library of integer stacks

Each learning section consisted of receiving the classroom lecture, observing the program behavior with TEDViT, practicing with a coding exercise, and receiving an explanation of the solutions for the exercise. We planned to allow the participants to use TEDViT mainly for observing program behavior so that they could observe and compare the memory image of variables and the target domain world by operating individually the learning environment visualized by TEDViT. We also observed that the teacher used TEDViT to explain the point of an exercise solution with connections between the status of the target domain world and the memory image of variables.

The teacher prepared the target program codes observed by the participants with TEDViT in the practice and the configuration files for the visualization of the target domain world. The number of prepared teaching materials (i.e., program codes and configuration files) was ten. The visualizations defined by the teacher in accordance with the teacher's own intent included arrow objects for representing the relationships between pointers and pointed variables, and coloring to highlight the objects on which the teacher intended the participants to focus. Figure 6 provides an example of the actual learning environment in the practice. Here, we extended the visualization of the memory image in TEDViT with the following functions:

- Visualizing the variable size and the size of the data pointed by the variable if it is a pointer.
- Visualizing the memory table where the row height corresponds to the variable size.
- Highlighting the operand variable in the address operation if the focusing statement includes an address operation.

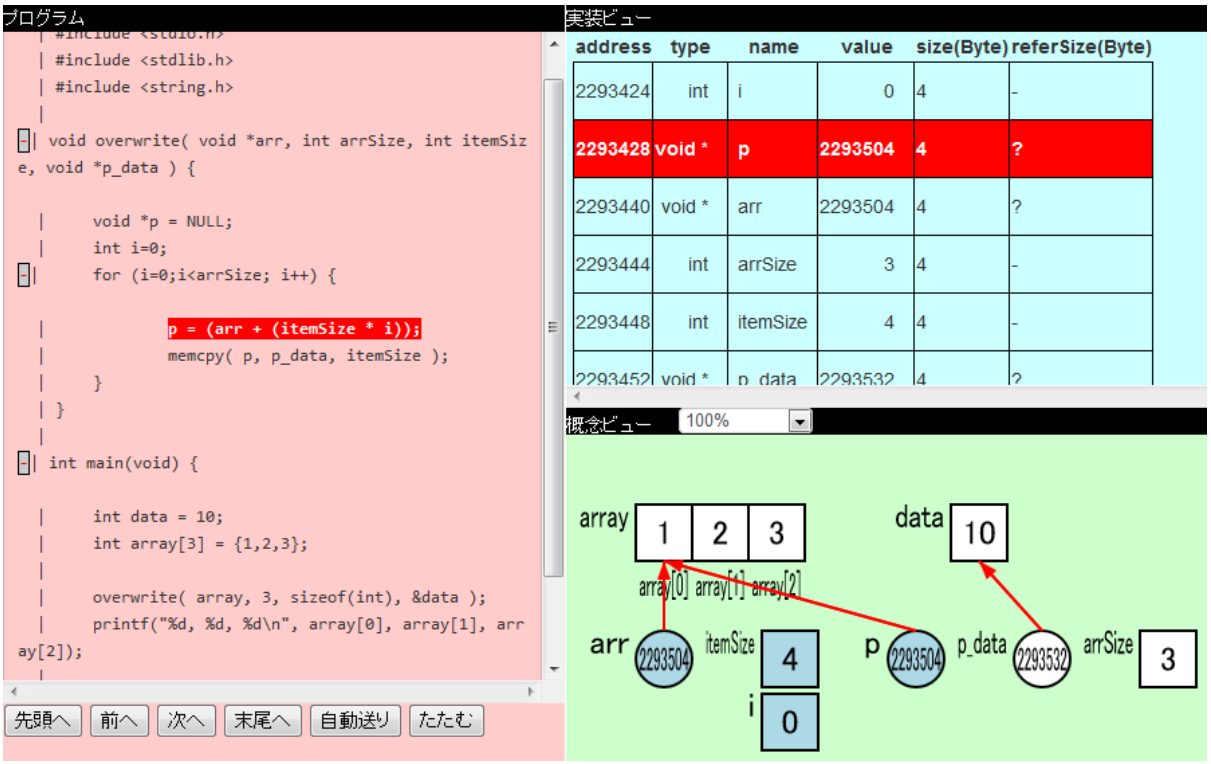

Figure 6. Example of environment for learning generic pointers.

3.2 Classroom Practice Evaluation

It was difficult to conduct a certain test for the evaluation of the learning effect with TEDViT because the participants in the practice were not students, but engineers already employed. Hence, our evaluation of the classroom practice is based on an analysis of the answers of two questionnaires conducted after the practice. One of them (Q1) was about learning with TEDViT, and the other (Q2) was about the learning content in the session. The former contains six items.

Q1-1. Asks how much TEDViT contributed to understanding C programming using five-point scale.
Q1-2. Asks how much TEDViT contributed to ascertaining wanted matters using five-point scale.
Q1-3. Asks how much TEDViT was needed for learning using five-point scale.
Q1-4. Asks whether the teaching materials are regarded as useful from among the prepared ten materials (allows multiple answers).
Q1-5. Asks for comments on the advantages and disadvantages of learning with TEDViT.
Q1-6. Asks for comments on the insufficient or inconvenient functions of TEDViT.

Table 5 provides an average of the answer scores for Q1-1, Q1-2, and Q1-3. These scores suggest that the participants were basically satisfied with the learning support provided by TEDViT. Table 6 lists the teaching materials found useful by a few participants (Q1-4). We consider that the indicated materials are divided broadly into two categories: 1) learning items where a learner is required to grasp the relationship between the target domain world and memory model, such as "pointer operators and memory model," "calls by pointer," and "operations on generic pointers;" 2) items with complicated behavior, such as "handling an array to reverse the order" and "sample library of integer stacks." These favorable evaluations for the two categories of our teaching materials suggest that our aim of supporting learners in understanding pointers in connection with a memory model using TEDViT was achieved successfully to a certain degree.

Table 5: Average and variance of answered scores in Q1-1, Q1-2, and Q1-3.

	Average	Variance	SD
Q1-1	4.60	0.24	0.49
Q1-2	4.40	0.37	0.61
Q1-3	4.33	0.36	0.60

Table 6: Number of answers indicating useful teaching materials (multiple replies allowed).

Teaching material	N
Pointer operators and memory model	7
Calls by value	3
Calls by pointer	7
Return values with pointers	4
Relationship between array names and pointers	3
Relationship between arithmetic operations on pointers and array index operations	3
Memory model of multidimensional arrays	4
Handling an array to reverse the order	5
Operations on generic pointers	9
Sample library of integer stacks	9

The latter questionnaire (Q2) has five items, all of which are similar to those conducted over several years at the HEPT training course:

Q2-1. Asks how much the participant was interested in learning the session content using five-point scale.
Q2-2. Asks how well the participant understood the session learning content using five-point scale.
Q2-3. Asks how fast the participant perceived was the teacher's progress in the session using five-point scale (1 = too slow, 5 = too fast).
Q2-4. Asks how difficult the session learning content was using five-point scale (1 = too easy, 5 = too difficult).
Q2-5. Asks how difficult the exercises were in the session using five-point scale (similar to Q2-4).

Table 7 provides the average and variance of each answer score in Q2, including those in the questionnaire conducted in 2013 and 2014. The corresponding learning sessions without TEDViT in 2013 and 2014 were conducted by the same teacher from our practice. We can observe that the variance of answer scores on the understanding degree (Q2-2), session speed (Q2-3), and session difficulty (Q2-4) with TEDViT are lower than those without TEDViT. There are no participants in our practice who answered "not understandable at all" or "slightly not understandable" in Q2-2, in contrast to the sessions in 2013 and 2014, where some participants did provide these answers. These reductions in variance suggest that our practice with TEDViT contributed to reducing irregularities in the participants' understanding, regardless of the wide variety of background knowledge, as indicated in Table 3. We consider that the reason for this is that learning support reached every participant using TEDViT.

Table 7: Average and variance of answered scores in Q2.

	2015 (using TEDViT)			2014			2013		
	Average	Variance	SD	Average	Variance	SD	Average	Variance	SD
Q2-1	4.67	0.36	0.60	4.31	0.34	0.58	4.33	0.82	0.91
Q2-2	4.07	**0.60**	0.77	4.38	0.86	0.93	3.17	1.21	1.10
Q2-3	3.07	**0.20**	0.44	3.06	0.43	0.66	3.37	0.30	0.55
Q2-4	3.27	**0.33**	0.57	3.13	0.73	0.86	3.53	0.58	0.76
Q2-5	3.47	0.52	0.72	3.25	0.69	0.83	3.60	0.44	0.66

4. Conclusion

In this paper, we described a learning support system called TEDViT, and the classroom practice for understanding pointers where we introduced our system. There are many existing systems that visualize the data structures processed by target programs in certain ways, and support learners in understanding programs and algorithms by making their behavior visible. However, concrete values of pointer variables are often concealed in such visualizations, and hence the way in which each visualized object is located on the memory is not made explicit. Therefore, novice learners often fail to grasp the relationship between program behavior and program code, and reach a learning impasse.

To address this issue, TEDViT visualizes simultaneously and synchronously the memory image that is the field for presenting the concrete values of the variables, and the target domain world that is the field for presenting logically the data structures processed by the target program. Moreover, in order to allow teachers to set their instruction content based on the growing variety of learner background knowledge, TEDViT visualizes the status of the target domain world according to the visualization policy defined by the teacher. Using TEDViT, teachers can provide flexible visualizations that are consistent with the teacher's class instructions, and learners can be less confused with the visualizations.

We conducted a classroom practice for understanding pointers in connection with a memory model. The participants in this practice were not university students, but software engineers responsible for software development. The participants learned program behavior by observing and comparing the memory image and target domain world visualized in the learning environment of TEDViT. In the questionnaire conducted after our practice, we obtained generally satisfactory answers for TEDViT. The analysis of answered scores suggested that TEDViT contributed to reducing the irregularities of the participants' understanding, and that appropriate learning supports reached every participant using TEDViT. In the questionnaire, some participants commented that the observations of the memory image in learning pointers were especially valuable. The comments by the teacher of our practice were also positive in that introducing TEDViT to the class would have a certain degree of effectiveness in understanding pointers.

Based on the educational practices that we have conducted thus far, including the practice described in this paper, we consider that visualizing synchronously the memory image and target domain world has a certain effect in learning by observing program behavior, and that the usability of TEDViT can be introduced to actual classrooms. We also consider that the definition method of the

visualization policy implemented in TEDViT has sufficient capability for visualizing a wide variety of teacher instruction intents.

At the time of the practice described in this paper, the teacher had to edit directly the configuration file with application software, such as Excel, in order to describe the drawing rules of the status of the target domain world when defining the visualization policy. We are currently developing a GUI tool for describing the drawing rules more intuitively. In future work, we plan to reduce the cost of preparing teaching materials with it. Furthermore, we will conduct more educational practices and evaluate the effectiveness of using TEDViT with higher reliability.

Acknowledgements

This study was supported by JSPS KAKENHI Grant Numbers JP24300282, JP16K01084.

References

Fossati, D., Eugenio, B. D., Brown, C., & Ohlsson, S. (2008). Learning linked lists: Experiments with the iList system. *Proceedings of the 9th International Conference on Intelligent Tutoring Systems*, 80-89.

Kogure, S., Fujioka, R., Noguchi, Y., Yamashita, K., Konishi, T., & Itoh, Y. (2014). Code reading environment according to visualizing both variable's memory image and target world's status. *Proceeding of the 22nd International Conference on Computers in Education*, 343-348.

Malmi, L., Karavirta, V., Korhonen, A., Nikander, J., Seppälä, O., & Silvasti, P. (2004). Visual algorithm simulation exercise system with automatic assessment: TRAKLA2. *Informatics in Education, 3*(2), 267-288.

Moreno, A., Myller, N., Sutinen, E., & Ben-Ari, M. (2004). Visualizing programs with Jeliot3. *Proceedings of the working conference on advanced visual interfaces*, 373-376.

Naps, T. L., Rößling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., Korhonen, A., Malmi, L., McNally, M., Rodger, S., & Velázquez-Iturbide, J. (2002). Exploring the role of visualization and engagement in computer science education. *ACM SIGCSE Bulletin, 35*(2), 131-152.

Neve, P., Hunter, G., Livingstone, D., & Orwell, J. (2012). NoobLab: An intelligent learning environment for teaching programming. *Proceedings of the 2012 IEEE/WIC/ACM Joint Conferences on Web Intelligence and Intelligent Agent Technology, 3*, 357-361.

Pears, A., Seidman, S., Malmi, L., Mannila, L., Adams, E., Bennedsen, J., Devlin, M., & Paterson, J. (2007). A survey of literature on the teaching of introductory programming. *ACM SIGCSE Bulletin, 39*(4), 204-223.

Yamashita, K., Fujioka, R., Kogure, S., Noguchi, Y., Konishi, T., & Itoh, Y. (2015). Educational practice of algorithm using learning support system with visualization of program behavior. *The 23rd International Conference on Computers in Education*, 632-640.

Yamashita, K., Nagao, T., Kogure, S., Noguchi, Y., Konishi, T., & Itoh, Y. (2016). Code-reading support environment visualizing three fields and educational practice to understand nested loops. *Research and Practice in Technology Enhanced Learning, 11*(1), 1-22. doi:10.1186/s41039-016-0027-3