Liu, C.-C. et al. (Eds.) (2014). Proceedings of the 22nd International Conference on Computers in Education. Japan: Asia-Pacific Society for Computers in Education

An Educational Practice Using a Code Reading Support Environment for Understanding Nested Loop

Koichi YAMASHITA [a*], Takamasa NAGAO[b], Satoru KOGURE[b], Yasuhiro NOGUCHI[c], Tatsuhiro KONISHI[b], & Yukihiro ITOH[d]

[a]*Faculty of Business Design, Hamamatsu 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 code reading support environment and a classroom practice using our system for understanding of nested loop. In our preceding work, we had practiced an exercise class in nested loop with a code reading support environment. The evaluation results suggested that students could obtain an expected learning effect roughly by using our system. However, we also found some of them had reached a learning impasse in the classroom. We tried to cope with them, based on two supporting approaches; to bridge a gap between generalization structures of program code and corresponding operations, and to make students capable of predicting the behavior of the nested loop. We extended the preceding system with some new functions according to our approaches, and practiced renewed exercise class in nested loop with our system. The evaluation results suggest that our new system has a certain correlation with understanding of nested loop.

Keywords: Education for programming, Domain world models, Learning environment for exercise, Education for informatics, Classroom practice.

1. Introduction

As an information level in our society advances, increasingly more various students need a computer programming education (Robins, Rountree, & Rountree, 2003; Pears et al., 2007). In our classroom experience, we have been attracted an attention by following three fundamental skills which novice programming students tend to feel difficult to acquire:

F1. Nested structure; some students are hard to understand it in control flow or code description.
F2. Generalization; some students are hard to generalize a set of concrete operations into an abstract function with some variables.
F3. Tracing; some students are hard to grasp values of variables clearly, which is changing through each statement.

Nested loop is a learning target which novice students would stumble for the first time, because learning of that needs to understand or to acquire above all three fundamentals. Koppelman and van Dijk (2010) emphasized the importance of nested loop as one of the targets to understand the concept of abstraction. However, limited time of the course would hardly allow students to get a deep understanding of these fundamental concepts. In order to have our students learn them efficiently, we have tried to introduce learning support systems into our classroom exercise in nested loop (Kogure et al., 2013).

Generally, programming students learn algorithms, code-readings and codings in turn. We have already developed the learning support system for the code-reading stage (Kogure, Okamoto, Noguchi, Konishi, & Itoh, 2012). As in our preceding work, we assumed that learners understand programs and algorithms by having an image consisting of three fields in their minds: program-code, objects processing by the program (target domain world), and sequence of concrete operations for the target domain. Learners have to grasp the relationships and correspondences among the components in each

field there. Our preceding system supports to understand a program-code by visualizing three fields and their relationships.

In our preceding work, we practiced exercise classes in nested loop with our system and found some students had reached a learning impasse in the classroom. Based on implicit and explicit feedback from our students, we constructed two approaches to cope with the problems; to bridge a gap between generalization structures of program code and corresponding operations, and to make students capable of predicting the behavior of the nested loop by observing two characteristics in the code.

In this paper, we describe new functions incorporated to our system according to these approaches and renewed classroom practice for nested loop. The evaluation results suggest that our extended system has a certain correlation with understanding of nested loops. We describe the preceding system in Section 2, two approaches to avoid the impasse in Section 3, an overview of our extended system in Section 4, respectively. We provide an overview of our classroom practice, our controlled experiment, and evaluation result of our system in Section 5, and we conclude with a brief summary and discussion of future work in Section 6.

2. Learning Environment for Programs and Algorithms

2.1 Our Preceding Work

In our preceding work, we assumed that learners need to have a image consisting of three fields and to grasp the relationships among their components; program-code field (PF), target domain field (TDF), and operations field (OF). Under this assumption, we developed the system which supports students to understand programs. Figure 1 shows the overview of learning support environment provided by the system (hereafter called LEPA). Each of three fields is reproduced in (A), (B) and (C), respectively.

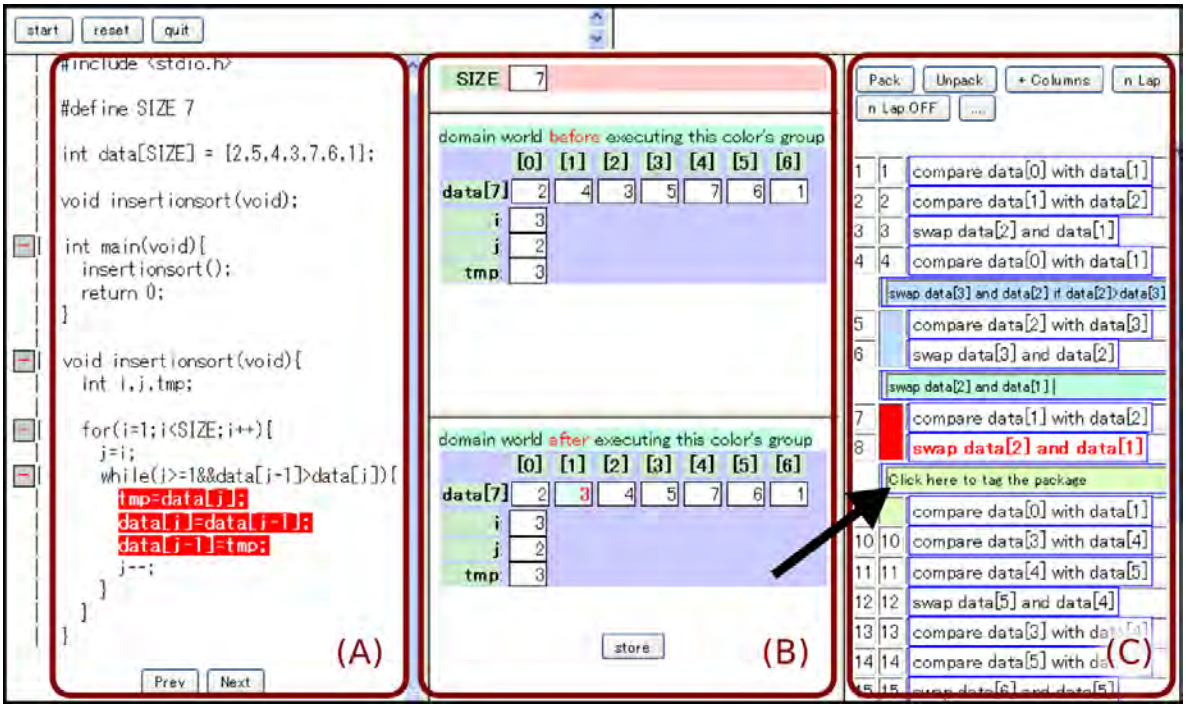

Figure 1. Overview of the environment provided by LEPA

A Learner can click one of operations in (C), so that the system displays the state of target domain after executing it in (B). S/he can know the role of a certain sequence of operations by comparing or observing visualized TDF before and after executing that. The system highlights the code fragment in (A) corresponding to the selected operation in (C), and vice versa because the system also allows him/her to click a fragment in (A). S/he can know the correspondence between a code fragment and an operation. We believe that s/he can store many pieces of intelligent information from

visualization by LEPA: what may happen in TDF by executing a concrete operation, or what kind of the code is needed to cause an effect on there.

Learning with LEPA is based on learner's externalization of his/her stored information. The externalization can be done by his/her packing and tagging process using GUI interface. If s/he found a certain sequence of operations having single abstract function in (C), then s/he could use GUI interface of the system in following two steps:

1. S/he can push “pack” button to pack the selected operations sequence into a package. Nested structure of packages is allowed.
2. S/he can tag the package with a natural language description according to its function (as in pointed part in Figure 1).

The resultant packing structure of operations sequence gets closer to the program code structure, ideally. We consider s/he will get to understand whole control sequences of the program code in the overall process of a series of these activities.

An activity of externalization can be classified into two classes according to the repetitiveness in target operations: abstraction for operations without that, generalization for ones with that. LEPA supports the former activity by the function of packing operations, and the latter by the one of tagging *n*-th lap of a loop (as in Figure 2).

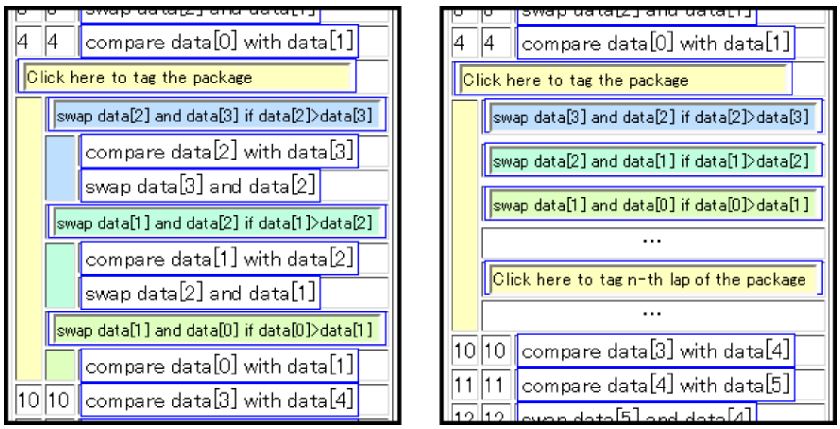

Figure 2. Tagging a package and tagging *n*-th lap of a package

2.2 Related Works

Shneiderman, Mayer and Heller (1977) defined a flowchart to be represented a high level definition of the solution to be implemented on a machine. Based on that, they stated flowcharting and programming can be separable independent tasks. It suggests that we may treat elaborating an algorithm and writing a program code as two separated tasks. We think that understanding an algorithm and a program code also can be separable.

So far, several intelligent tutoring systems are developed to support programming learners. They include RoboProf (Daly & Horgan, 2004), JITS (Sykes & Franek, 2003), J-LATTE (Holland, Mitrovic, & Martin, 2009), BITS (Butz, Hua, & Maguire, 2006), and so on. Several learning support systems based on algorithm visualization are also paid a lot of attention, including TRAKLA2 (Malmi et al., 2004), Jeliot3 (Moreno, Myller, Sutinen, & Ben-Ari, 2004; Čisar, Pinter, Radosav, & Čisar, 2011), ViLLE (Rajala, Laasko, Kaila, & Salakoski, 2008), and so on. Classifying these systems from the standpoint of the tasks understanding an algorithm and a program-code, the main target in every system seems to support either of them. Our attractive target is a gap between two tasks. We consider that these systems will provide insufficient support to bridge the gap.

Learners who have a proper understanding of an algorithm can reproduce its behavior for a concrete data. A sequence of operations in LEPA is a sequence of natural language descriptions representing the algorithm behavior. Hence, operations sequence can be regarded as an externalization of his/her understanding of the algorithm. Some other existing systems visualize relationships between a program code and its target domain world. LEPA also does it, and furthermore, relationships between a sequence of operations and its target domain. It also visualizes the correspondence relationships between a code fragment and an operation. We expect that these visualizations of three fields and

relationships among them contribute to bridging the gap between two tasks. That is why we base our approaches to bridge the gap on LEPA.

Given a program comprehension task to a programmer, his/her procedure for code reading consists of two steps normally; recognizing the function of groups of statements, and then piecing together these chunks to form ever larger chunks (Shneiderman & Mayer, 1979). S/he continues these steps hierarchically until the entire program is comprehended. LEPA has a function to support a learner to behave oneself like that in OF. Therefore, users can learn the procedure for code reading. However, if the chunk corresponds to iteration like a loop, it is often the case that his/her recognition involves generalization with some variables. As described below, LEPA supports insufficiently to generalize an iterative package.

We can find some systems targeting nested loop, including AlgoTutor (Yoo, Yoo, Seo, Dong, & Petty, 2012), the tutor developed by Dancik and Kumar (2003), and so on. However, it is not evident how they lead learners to understand or to acquire the fundamental concepts F1, F2, and F3 described in Section 1. Our works described in this paper aim to sophisticate the learning supports in LEPA by elaborating the strategy of hierarchical procedure for code reading.

3. Our Supporting Approaches

In our preceding work, we had practiced an exercise class in nested loops with LEPA. In that class, we found some of the students had reached a learning impasse. Based on the score differences between pre/post tests, the system did not have statistically significant correlation with student's ability of generalization. In this section, we describe two approaches to tackle them.

3.1 A Gap between the Representations of Generalization

Packing the operations is taking them closer structurally to the program code. However, we can find a gap between the structure of a package and corresponding fragment of program code as in Figure 3.

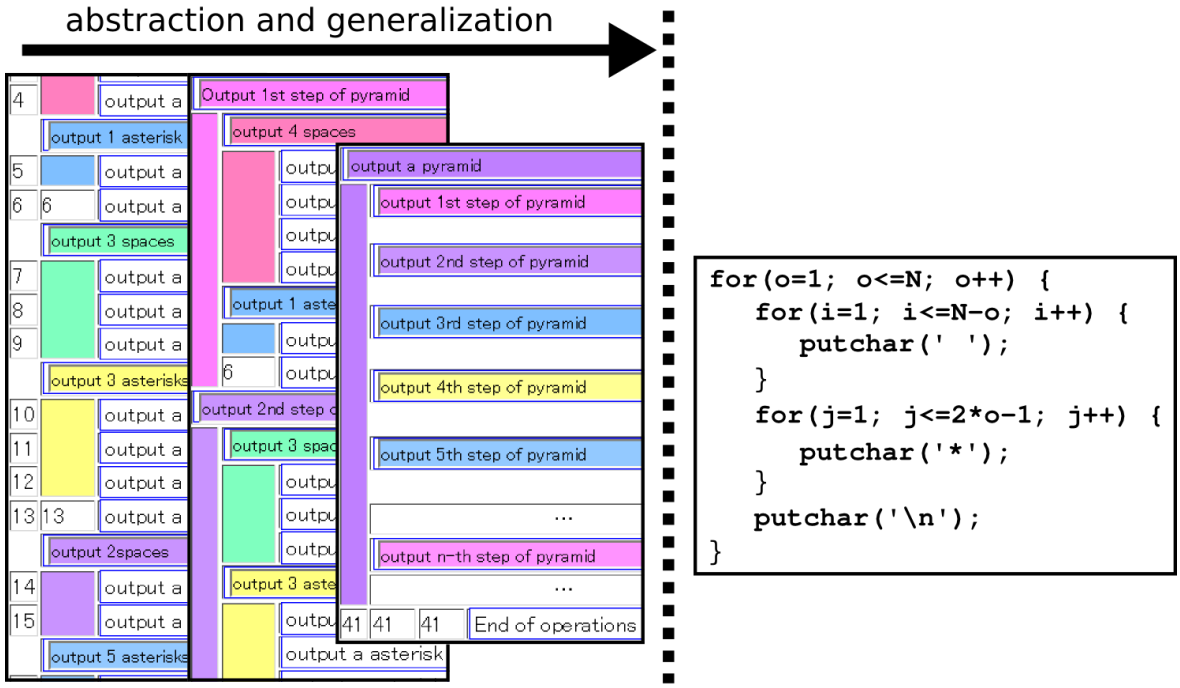

Figure 3. A gap between generalized operations and program code

The right side of Figure 3 shows a program code which displays a *N*-step pyramid by outputting an adequate number of spaces, asterisks, and new-lines. Other side shows a transition of the package structure corresponding to the code. This would be the most possible transition, we think.

We expect those packages are made through following steps of a student's learning: First, s/he observes three fields, and consequently makes two packages like "output *x* spaces" and "output *y* asterisks". Here, s/he assigns concrete numerics in *x* and *y*, not variables. Then, s/he packs hierarchically them plus an operation which outputs a new-line, and tags it "output *z*-th step of pyramid". *z* is a concrete numeric, again. S/he continues these steps until all operations in OF are included in packages. Finally s/he gets *N* packages, packs them, and tags *n*-th lap.

In LEPA, the target of a generalization tagging is the first hierarchical level only. It means that s/he tags *n*-th lap with a variables only on the packages "output *z*-th step of pyramid". The system hides deeper levels than it in OF (outputting spaces, asterisks and new-lines). However, a program code of nested loop is given all over the hierarchy. Generalized the first level hierarchy into "output *k*-th step of pyramid" with variable *k*, then s/he should also do deeper levels as "output *N*-*k* spaces" or "output 2*k*-1 asterisks". With the generalizations throughout the hierarchy, s/he should realize the meanings of control statements for nested loops in the program code.

Based on these discussions, we have implemented a function to enable users to generalize a package as keeping the hierarchy explicit, so that s/he can tag *n*-th lap of inner loops. In addition, we have also implemented one to generate a template of general tag automatically in following steps;

1. discriminating variable words from invariable ones in the set of tags on packages to be generalized,
2. and replacing variable ones to a series of some symbol like "_".

For example, in displaying pyramid, our system generates templates like "output XXX spaces", "output ??? asterisks" and "output ___th step of pyramid" from the set of tags. Proposing templates to the learner, our system encourages him/her to formulate "XXX", "???", "___" with loop control variables in the program code. The screenshot in Figure 4 shows the implemented functions. They are expected to lead learners to understand the structure of nested loops.

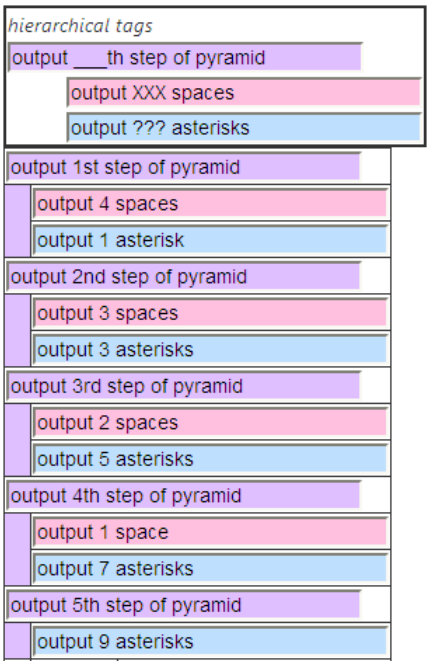

Figure 4. Generalization of packages as keeping the hierarchy explicit

3.2 Two Characteristics in Nested Loop Code

Investigating textbooks and exercises, we have classified the behaviors of nested loops learned by students into 4 types. The classification is based on following two characteristics in nested loop code:

Ch1. A statement to iterate in inner loop has a reference to control variables of the outer loop.
Ch2. A conditional statement of inner loop has a reference to control variables of the outer loop.

The complication of behavior of nested loop tends to grow with them, in the order of neither of them, Ch1, Ch2, and both of them. Learning these relationships between the behavior and the characteristics could enable learners to get a deep understanding of nested loop.

A programmer implements loops including not only the identical operations, but also wide variety of repetitive operations by referencing the loop control variables appropriately. For example, different values are displayed by iterative executions of the statement to display the value of variable *i* in a loop controlled by *i*. Therefore, the iteration package corresponding to the statement consists of a sequence of operations with different descriptions each other. The generalization with our system has two phases; to find operations varying regularly in an iterative package in OF, and to formulate the regularity.

A nested loop increases the diversity of repetitiveness in the corresponding operations. We have to take account of not only inner and outer loops in which their control variables are referenced respectively, but also loops with above two characteristics. A nested loop with Ch1 has operations with varying representation in each iteration step of the outer loop, that is, each inner loop. For example, left side of Figure 5 shows a program code outputting different values according to each iteration step of the outer loop. On the other hand, a nested loop with Ch2 has the different number of operations corresponding to the inner loop according to every iteration step of the outer one. For example, right side of Figure 5 shows a program code where the different number of operations corresponds to the inner loop according to every iteration step of the outer one.

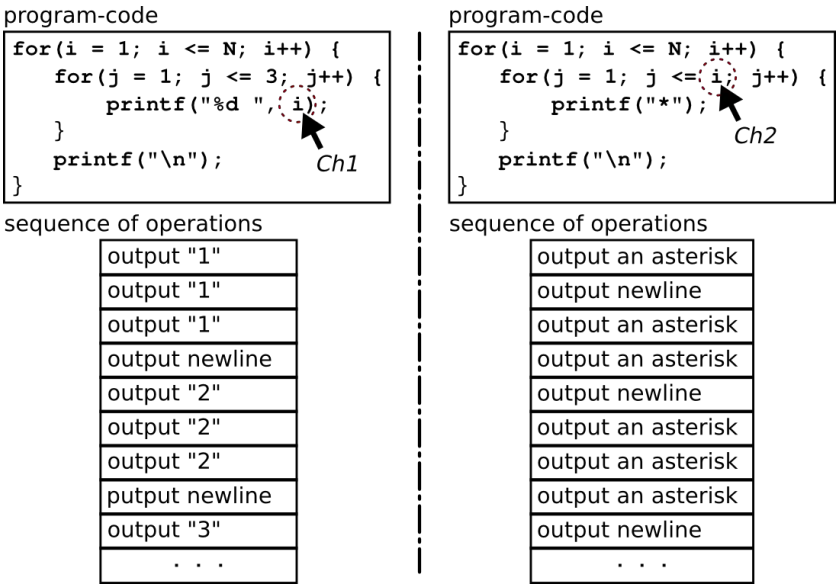

Figure 5. A program-code with Ch1/Ch2 and its operations sequence

We call the nested loops with Ch1 to "step contents varying type" of nested loops, and those with Ch2 to "step times varying type". A learner needs to recognize the repetitiveness of the operations sequence appropriately with or without these characteristics, in his/her series of packing. Furthermore, to recognize the repetitiveness, a learner needs to anticipate the regularity in OF according to characteristics in PF.

Based on these discussions, we have planned to teach students these characteristics, aiming to allow them to cultivate a better understanding. We have also implemented another function to support to learn them, which asks for an answer to the question; which type of nested loop does the one given by our system have; step contents varying type, step times varying type, or both types? For the students who cannot answer, our system gives a following sequence of stepwise hints:

1. Hints on which part of the program-code they should focus.
2. Hints about what characteristics they should read from the focusing part.
3. Characteristics that should have been read.

4. Our Extended System

4.1 Overview of Our Extended System

We show an overview of the environment provided by our extended system in Figure 6. Our system places more emphasis on supporting to learn nested loops than LEPA. Our system leads learners to learn based on following scenario of nested loop learning.

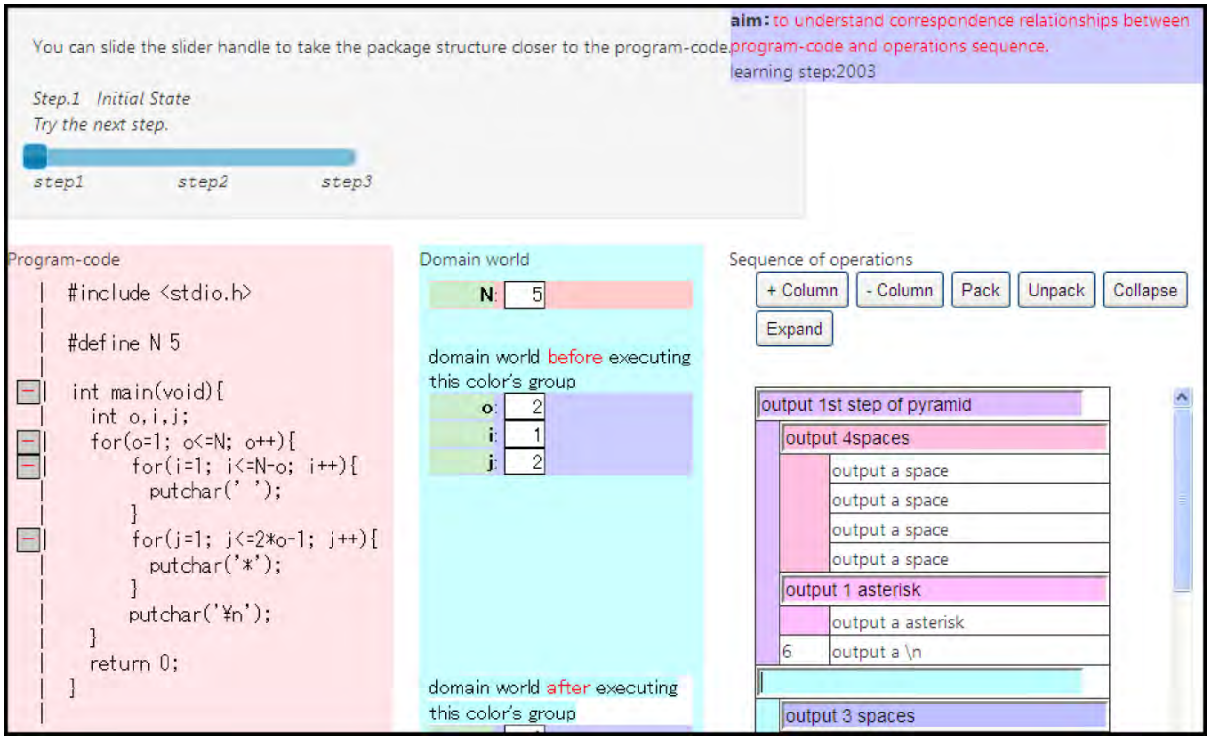

Figure 6. Overview of the environment provided by our system

Ex1. Tracing the program code to understand the behavior and control flow over the entire program.
Ex2. Packing the sequences of operations to recognize abstract functions and to understand the entire program as a function hierarchically.
Ex3. Generalizing the packages of operations sequences to understand the structures of the nested loop.
Ex4. Observing the characteristics of nested loop on the program-code to cultivate a better understanding of nested loops.

In nested loops, the repetitiveness to find in a generalization phase appears not on the concrete operations but on the tags on packages packed by the learner. We have implemented following functions to bring it to his/her attention in Ex3 above.

- Autocomplete function to support to make the repetitive packages by displaying the learner's tagging history.
- Function to support to focus on repetitive packages by setting similar tags to the same color.
- Function to support to look up a repetitiveness on the tags by hiding concrete operations.
- Function to support to tag *n*-th lap appropriately by providing the template of generalization tag.

4.2 Expected Learning Effect

Preceding educational practice suggests that learning with the environment visualizing three fields and relationships among them has a rough correlation with understanding or acquiring fundamentals F1 and F3 described in Section 1. In addition to these effects, our system could bridge the gap between each representation of generalization in the program code and a sequence of operations, and could lead learners to better understandings about nested loops. Consequently, it could also lead them to understandings of F2. We have hypothesized following two learning effects of our extended system.

Hypo1. Functions proposed in Section 3.1 could promote learner's understandings of behaviors and structures of nested loop.
Hypo2. Functions proposed in Section 3.2 could promote learner's understandings of relationships between implementations and behaviors of nested loop.

5. Evaluation of Learning Effects Based on Educational Practice

5.1 Overview of Classroom Practice

For the purpose of verification of two hypotheses described above, we practiced two consecutive exercise classes in nested loop with our extended system. Our classes were incorporated into a series of actual classes being held in department of administration and informatics, Hamamatsu University. The department holds two courses “Programming I” and “Programming II” for second year students. Our classes were practiced in the latter. The number of attending student was 12. Each of the students is majoring in business administration and had less than a year of learning programming.

In the first class, a teacher who regularly teaches the course gave a lecture on single and nested loops in 60 minutes as refresher training. The lecture includes the characteristics of nested loop described in Section 3.2. At the end of the class, we conducted a 15 minutes pretest to judge the students' understanding level of nested loops before using our system. In the second class, we allowed the students to use our system to learn nested loop. The program for this exercise is the one that displays a 5-step pyramid with spaces, asterisks, and new-lines. Before the exercise in the class, the teacher described the aims of the exercise and the environment provided by our extended system. During the exercise, neither the teacher nor we provided help to the students in understanding the program. After the 60 minutes exercise, we conducted a 15 minutes posttest to judge the students' understanding level of nested loop after using our system. We used BB FlashBack Express 3 (available on: http://www.bbsoftware.co.uk/BBFlashBackExpress.aspx) for recording the screen videos of students' interactions with the environment.

The programs used in the pretest and the posttest are different; however, the questions are almost the same. Question 1 (Q1) asks the execution result expected by tracing the entire program including nested loop by hand. Question 2 (Q2) asks the code fragment in the control of the inner-loop with the execution result given, and Question 3 (Q3) asks that of the outer-loop. Question 4 (Q4) asks the characteristics of nested loops to be expected to appear on program-code, given the execution results only. Q1, Q2, and Q3 are designed to verify the Hypo1, and Q4 for the Hypo2.

For the purpose of more detailed verification of advantage of learning with our system, we conducted another controlled experiment with 5 students as the control group. They are the second year students in the same course as the experimental group, but have not attended in our practiced class. First, they were given the same lecture on loop as experimental group in 60 minutes, including the characteristics of nested loop, and then we conducted the 15 minutes pretest. Next, they did the self-study with textbooks and lecture materials, never using our system. Before the self-study, the teacher described the aim of that is to understand nested loop with an example program. The example program is the same as one in the exercise practiced. The teacher also described the direction for self-study should be based on tracing the program. After the 60 minutes self-study, we conducted the 15 minutes posttest. The pre/post test are the same as ones for the experimental group.

Table 1 shows the differences of the average score on each question between pretest and posttest of experimental and control group. We marked each question in both test as following; Q1 out of 3, Q2 out of 9, Q3 out of 3, and Q4 out of 48. A tendency can be seen in that experimental group grows the marks significantly between the pre/post test but the control one does not.

Table 1: The differences of the average score between pre/post-test

	Q1	Q2	Q3	Q4
experimental	0.83	0.75	0.50	1.50
control	1.20	0.20	-0.60	-5.70

We think that more progress in control group on Q1 is caused by the direction for their self-study provided by the teacher. They could have consumed much of their time in tracing the program. Therefore, we consider they have an advantage over experimental group whose learnings consisted of 4 stepwise exercises.

5.2 Discussion

We examined carefully footage recorded for each student of experimental group in order to analyze learning effects in more detail. Consequently, we found the interactions with the learning environment provided by our system differ significantly among each of the students. For following 4 activities, we categorized the students into those who performed the activity (positive group) and those who did not perform the activity (negative group).

A1. The student consumes enough time in tracing the program (Ex1).
A2. The student packs all the operation sequences corresponding to the inner loops and tags all the packages (Ex2).
A3. The student accepts the guidance function of outer loop generalization appropriately (Ex3).
A4. The student uses the function to support to observe the characteristics of nested loop (Ex4).

We consider these activities lead the learning effect as follows.

- A1 to be the action to understand the behavior of the entire nested loop to acquire tracing skills (F3).
- A2 to be the action to understand the step contents of the inner loops and to understand the structure of the nested loop (F1).
- A3 to be the action to understand the step contents of the outer loop and to acquire the skills to generalize concrete operations (F2).
- A4 to be the action to understand the characteristics of the nested loops and to understand the relationships between the behavior and the characteristics on the program-code.

We expect the students performed these activities to grow their marks; the student performed A1 would grow his/her mark on Q1, the one performed A2 would do that on Q2, likewise the one performed A3 and A4 would do that on Q3 and Q4, respectively.

Table 2 shows the differences of the average score on each question between pretest and posttest of positive and negative group from A1 to A4. By performing an independent t-test, if the difference between negative and positive is statistically significant progress at the level $p = 0.05$, we put an asterisk next to the number. Each positive group shows significant progress their marks on corresponding question on the whole as expected.

Table 2: The differences of the average score in positive and negative group

		Q1	Q2	Q3	Q4
A1	Positive	1.43*	1.57	1.29*	
	Negative	0.00	-0.50	-0.75	
A2	Positive	0.80	2.60*	0.60	
	Negative	1.00	-0.67	0.50	
A3	Positive	1.17	1.67	1.50*	
	Negative	0.60	-0.20	-0.60	
A4	Positive				2.63*
	Negative				-0.75

The progresses in Q1, Q2 and Q3 suggest to favor the Hypo1, and that in Q4 suggests to favor the Hypo2. Hence, the results suggest that our system has a certain correlation with the understandings about nested loops on condition that user performs the expected activities.

6. Future Work

In this paper, we described the classroom practice for understanding of nested loop and the learning support system adopted there. Nested loop is an appropriate target to learn the fundamental skills of programming. We think that learning support systems contribute to understand or acquire them efficiently and effectively.

In the preceding educational practice with LEPA, we found some of the students had reached a learning impasse. We tried to cope with them, based on two supporting approaches; to bridge a gap between generalization structures of program code and corresponding operations, and to make students capable of predicting the behavior of the nested loop by observing two characteristics in the code.

According to them, we developed the function to generalize packages as keeping the hierarchy explicit for the former, and the function to support to observe the characteristics of nested loop for the latter.

We evaluated the effect of our system added the functions above in the actual classes. We also conducted a controlled experiment with 5 students as control group to verify the advantage of learning with our system in more detail. The evaluation results based on the scores on pre/post-test suggest that learnings with our system have a certain correlation with the understandings about nested loops.

As described in (Kogure et al., 2013), we have not only intended to support learners to learn nested loop. Our goal is to construct new form of education for programming with our code reading support environment. The correlation observed in the classroom practice suggests that our system will contribute to learner's understanding or acquisition of three fundamentals; F1, F2, and F3. We plan to collect more knowledge from more practices with our system, and to enhance our system and our classes.

Acknowledgements

This study was supported by Japanese Grant-in-Aid for Scientific Research (B) 24300282.

References

Kogure, S., Okamoto, M., Yamashita, K., Noguchi, Y., Konishi, T., & Itoh, Y. (2013). Evaluation of an algorithm and programming learning support features to program and algorithm learning support environment. *Proceedings of the 21st International Conference of Computers in Education*, 418-424.

Kogure, S., Okamoto, M., Noguchi, Y., Konishi, T., & Itoh, Y. (2012). Adapting guidance and externalization support features to program and algorithm learning support environment. *Proceedings of the 20th International Conference of Computers in Education*, 321-323.

Shneiderman, B., Mayer, R., Mckay, D., & Heller, P. (1977). Experimental investigations of the utility of detailed flowcharts in programming. *Comminications of the ACM*, 20(6), 373-381.

Shneiderman, B. & Mayer, R. (1979). Syntactic / semantic interaction in programmer behavior: A model and experimental results. *International Journal of Computer and Information Sciences*, 8(3). 219-238.

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and Teaching Programming: A Review and Discussion. Computer Science Education, 13(2), 137-172.

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.

Dancik, G. & Kumar, A. (2003). A Tutor for Counter-controlled Loop Concepts and Its Evaluation. Frontiers in Education, 1, T3C 7-12.

Yoo, J., Yoo, S., Seo, S., Dong, Z., & Pettey, C. (2012). Can We Teach Algorithm Development Skills? Proceedings of the 50th Annual Southeast Regional Conference, ACM, 101-105.

Koppelman, H. & van Dijk, B. (2010). Teaching Abstraction in Introductory Courses. Proceedings of the 15th annual conference on Innovation and technology in computer science, 174-178.

Daly, C. & Horgan, J. M. (2004). An automated learning system for java programming. *IEEE Transaction on Education*, 47(1), 10-17.

Sykes, E. R. & Franek, F. (2003). An intelligent tutoring system prototype for learning to program java. *Proceedings of the 3rd IEEE International Conference on Advanced Learning Technologies*, 485-492.

Holland, J., Mitrovic, A., & Martin, B. (2003). J-latte: a constraint-based tutor for java. *Proceedings of the 17th International Conference on Computers in Education*. 142-146.

Butz, C. J., Hua, S., & Maguire, R. B. (2006). A web-based Bayesian intelligent tutoring system for computer programming. *Journal of Web Intelligence and Agent Systems*, 4(1), 77-97.

Malmi, L., Karavirta, V., Korhonen, A., Nikander, J., Seppälä, O., and 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.

Čisar, S. M., Pinter, R., Radosav, D., & Čisar, P. (2011). Effectiveness of program visualization in learning java: a case study with Jeliot3. *International Journal of Computers, Communications & Control*, 6(4), 669-682.

Rajala, T., Laasko, M. –J., Kaila, E., and Salakoski, T. (2008). Effectiveness of program visualization: a case study with the ViLLE tool. *Journal of Information Technology Education*, 7, 15-32.