Wong, L.-H. et al. (Eds.) (2013). Proceedings of the 21st International Conference on Computers in Education. Indonesia: Asia-Pacific Society for Computers in Education

Evaluation of an Algorithm and Programming Learning Support Environment based on Classroom Practices

Satoru KOGURE[a*], Makoto OKAMOTO[a], Koichi YAMASHITA[b], Yasuhiro NOGUCHI[c], Tatsuhiro KONISHI[a] & Yukihiro ITOH[d]
[a]*Graduate School of Infomatics, Shizuoka University, JAPAN*
[b]*Faculty of Economics, Tokoha University, JAPAN*
[c]*Faculty of Informatics, Shizuoka University, JAPAN*
[d]*Shizuoka University, JAPAN*
*kogure@inf.shizuoka.ac.jp

Abstract: In this paper, we describe an algorithm and programming learning environment by illustrating the relationships among programs, target domains, and operations. Our proposed learning environment supports software programming learners in understanding code that includes nested loops by using a function that visualizes the differences in the target domain's state before and after executing a sequence of operations. We evaluate this environment in the classroom, and the results suggest that the proposed environment improves the understanding of programming beginners.

Keywords: Learning support environment, nested loops, teaching algorithm and programming

1. Introduction

Several research studies have focused on visualized programming learning environments (Butz *et al.* (2006), Fossati *et al*. (2008), Gabor (2009), Malmi *et al.* (2004) and Noguchi *et al.* (2010)). However, definitive educational methods have not yet been established for learning algorithms and programming, because learning units in these areas are very large in scope (McCracken *et. al.* (2001) and Bennedsen & Caspersen (2008)).

Ideally, items to be learned are divided in smaller chunks, and learners learn each chunk in three stages. The first stage is the algorithm-understanding phase (S1). Typically, learners attend a lecture (S1-1) and then reproduce the behavior of the algorithm using the specified input data (S1-2). Learners also produce operation sequences that represent the behavior of the algorithm. The second stage is the code-understanding phase (S2). Learners learn to understand the operation sequences and the state of the target domain (S2-1). They also understand the program code as a whole, based on the operation sequences (S2-2). The third stage is the coding phase (S3). Learners write (S3-1) and test (S3-2) their own program code.

We have already constructed the algorithm learning environment for the S1-2 stage (Noguchi *et al.* 2010), and we have also constructed the algorithm and programming learning environment for the S2 stage (Kogure *et al.* 2012). In this study, we focus again on the S2 stage. As in the previous study, we define three fields in programming: the programs field (PF), the target domain field (TDF), and the operations field (OF). It is important to understand these three fields and the relationships among them in order to understand programs and algorithms. PF stores the program code. OF stores the operation sequences. TDF visualizes the state of the target domain when an operation sequence is executed.

In the previous study, we suggested that it is important for learners to understand the relationship among PF, TDF, and OF on the S2 stage. Learners can understand the operation sequences for specific input data and the state of the target domain corresponding to the operation sequences with relative ease. However, they find it very difficult to understand the behavior of the

program code. One cause of the difficulty is the gap between the concrete fields (TDF and OF) and the abstract field (PF). Therefore, we propose a new method for understanding program code in order to bridge this gap for learners.

The system helps the learner learn through the following procedure: First, learners create a set of operation sequences to process specific input data. Then, learners label the set with its intended purpose. By doing so, the system promotes the learners' understanding of the relationship between OF and PF and between TDF and PF. However, it is difficult for programming beginners to understand certain concepts, such as nested loops and recursive functions.

The primary target of this paper is to focus on teaching multiple nested loops in programs, and we evaluate our previous environment (Kogure *et al.* 2012) for promoting this understanding in actual classroom in a university. We evaluate following three effects of adopting our proposed system in the classroom. The first effect is that the learners would understand nested loops (E1). The second effect is that the learners would be able to trace through a program that includes nested loops (E2). The third effect is that the teacher would be able to measure the learners' level of understanding by observing a learner's behavior in the system (G3). We asked a teacher to use our proposed system in a classroom setting, and the results suggested that all three effects were achieved.

2. Concept of Existing Algorithm and Programming Learning Environment

2.1 Three Fields of Programming

We believe that it is very important for learners to understand three important fields in programming and the relationships among these three fields. As mentioned earlier, these three fields are the programs field (**PF**), the target domains field (**TDF**), and the operations field (**OF**). **PF** manages the program code. **TDF** presents the state of the target domain, before and after the system executes certain operation sequences. **OF** manages the operation sequences. We explain the three fields in more detail.

Based on the learning goals, the teacher selects an algorithm and a program code to illustrate the algorithm. Then, the learners select the input data to use in order to check the behavior of the sample program. The sample program is stored in **PF**, where a fragment of the sample program p_i is included in program set $P = \{p_1, p_2, \ldots, p_i, \ldots, p_N\}$. The system automatically embeds a program fragment p'_i (typically a *sprintf* operation) to observe the behavior of a fragment p_i in the original program set P. The system compiles and executes the program set $P' = \{p_1, p'_1, p_2, p'_2, \ldots, p_i, p'_i, \ldots, p_N, p'_N\}$, then it obtains the execution histories $EH = \{eh_1, eh_2, \ldots, eh_j, \ldots, eh_M\}$. If P includes nested loops, the number of execution histories M is greater than the number of code fragments N. The system converts the execution histories into operation sequences, which reproduce the operation for the algorithm-understanding stage (S1-2); for example, an operation sequence could be "compare A with B" or "swap A for B" for the sort algorithm. Therefore, the set of operation sequences $OS = \{os_1, os_2, \ldots, os_k, \ldots, os_O\}$ is in **OF**. The system can present the states of a target domain, std_k after the system executes the operating sequence os_1 to os_k in order for the learner to analyze the effects of the operating sequences. Therefore, the set of the target domain's state, $STD = \{std_1, std_2, \ldots, std_k, \ldots, std_O\}$ are in **TDF**.

Our proposed algorithm and programming learning environment promotes the learner's understanding of the code by illustrating the relationships among the P in **PF**, the OS in **OF**, and the STD in **TDF**, caused by the execution of EH.

*2.2 Gap between the Abstract Field **PF** and the Concrete Fields **TDF** and **OF***

In the program P in **PF**, the teacher describes the algorithm at a high level. The OS in **OF** and STD in **TDF** are concrete forms, because the system generates OS and STD from EH, whereas the system executes P' for specific input data. Programming learners easily understand the behavior of OS and STD; however, they have difficulty understanding the behavior of P.

In this study, we use the following three processes to help learners understand a program that includes nested loops. First, the system help learners to understand the purpose of os_k, to observe std_k,

and to compare std_k with std_{k-1}. Second, the system teach learners how to insert an *os* (from os_k to $os_{k'}$) into a package pos_k, so that learners would be able to understand and describe pos_k and to compare $std_{k'}$ with std_{k-1}. A pos_k, which is described in a tag, corresponds to an inner loop in the sample program. Third, the system teach learners how to insert a *pos* and/or an *os* into a package pos'_k, so that learners would understand and describe pos'_k in order to space out each pos_k. This way, learners understand the outer loop. Thus, the system helps learners to fill the knowledge gap between the abstract field **PF** and the concrete fields **TDF** and **OF**.

2.3 Externalization to Illustrate the Learners' Understanding

In a programming exercise, the teacher asks the learners to describe their own understanding of the concepts. This task has two purposes. First, the learners are able to objectively reaffirm their own understanding. Second, the teacher can check each learner's level of understanding. Therefore, the system must have at least two functions. One is a function where the learners verbalize the intent of a set of operation sequences. Another is a function where the learners pack the operation sequences into a package using a GUI interface in order to illustrate the coding/abstraction phase.

3. Overview of Existing Algorithm and Programming Learning Environment

Figure 1 shows the interface of our proposed learning environment. In Figure 1, (1) corresponds to **PF**, (2) corresponds to **TDF**, and (3) corresponds to **OF**.

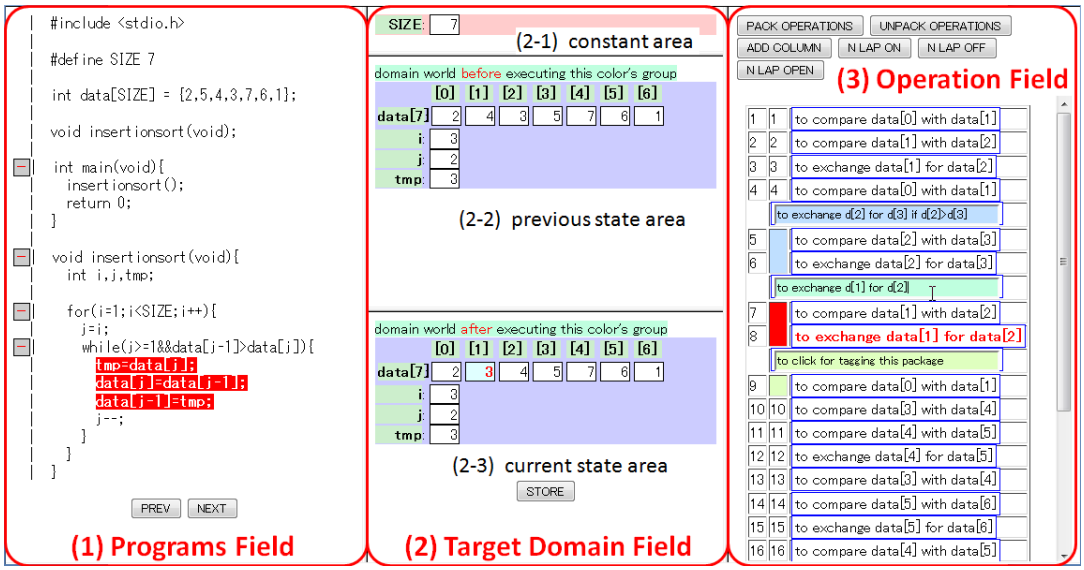


Figure 1. Interface overview.

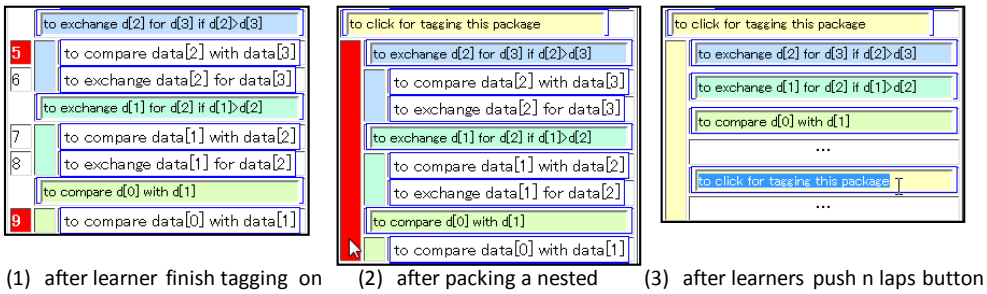
If learners push either the "PREV" button or the "NEXT" button, the system displays the previous execution history eh_{j-1} or the next execution history eh_{j+1}, respectively, and it highlights the program fragment p_i that corresponds to the execution history eh_j on **PF**. The system also helps the learner visualize the std_k that corresponds to the execution history eh_j on **TDF**, and it highlights the operation sequence os_k that corresponds to the execution history eh_j on **OF**. We believe that learners' program tracing skills are improved by observing the state of target field synchronously with executing step-by-step (E2 described in section 1).

TDF consists of the constant area (Figure 1 (2-1)), the previous state area (Figure 1 (2-2)), and the current state area (Figure 1 (2-3)). To compare the states before and after executing a certain program fragment p_i in **PF** or a certain operation sequence os_k in **OF**, learners can use the state storage function. If learners push the "STORE" button, the system copies the information of the current state area to the previous state area. By observing the differences between the previous state area and the current state area, learners learn the behavior of a specific statement or operation.

In **OF**, to generalize the operation sequences os_k, learners can pack any operation sequence as shown in Figures 1(3). Then, learners can label the group of operation sequences in Figures 1(3). Next, when learners click the vertically long cell for the package of operation sequences, the system shows the values of the variables before and after the execution of the operation sequences in the previous state area (Figure 1 (2-2)) and in the current state area (Figure 1 (2-3)), respectively.
The learner generalizes operation sequences by selecting some of the operations, packing them into a package and tagging a meaning to the package. We believe that those generalizing activities lead the leaner to focus on the hierarchical structure of the whole operations and hence lead to more deep understanding of nested loops (E1 described in section 1). We also believe that the teacher can observe the learner's understanding level based on the learner's selecting, packing and tagging results which are visualized by our system (E3 described in section 1).

To ensure that the learner enters the correct tags, the system offers a selection of tags and asks the learners to select the tag that is most similar to meaning tagged by learner. It would be ideal for the system to automatically judge the correctness of learners' tagging meaning by using semantic analysis; however, understanding a natural language sentence is difficult to achieve with high accuracy.

Next, the system helps learners to understand one step of the outer loop using the n-lap externalization function. For example, in Figure 2(1), learners grouped and labeled three operation sequences. In Figures 2(1) and (2), they created a group of operation sequences by clicking "five" for the fifth operation, "nine" for the ninth operation, and the "PACK OPERATIONS" button (shown in Figure 1(3)). If the learner clicks on the "N LAP ON" button (Figure 1(3)), the system displays three labels written by the learner and a new input field where the learner describes the general meaning of the n-lap descriptions. By performing this task, learners can clearly understand one step of the outer loop.



(1) after learner finish tagging on 1, 2 and 3 laps in inner loops. (2) after packing a nested package (3) after learners push n laps button

Figure 2. Tagging n Laps.

4. Testing the Learning Environment in a Classroom Setting

4.1 Purpose and Conditions

We evaluate our system to confirm three effects: E1, E2 and E3 described in section 1. This evaluation tests the following three hypotheses:

- Hypothesis 1: Packing operation sequences into a package and tagging the package have a beneficial effect in understanding the processes of nested loops. (**Hypo1**)
- Hypothesis 2: Illustrating the relationships among **PF**, **TDF** and **OF** has a beneficial effect on tracing a program that includes nested loops. (**Hypo2**)
- Hypothesis 3: The learning environment can be an effective means for the teacher to judge each learner's level of understanding. (**Hypo3**)

The subjects for the experiment were 30 art students (sophomores) from a university in Japan. Each of the subjects had less than a year of learning programming.

For the experiment, two consecutive lectures were given in the university. The first lecture about single loop and nested loops was given by a professor who regularly teaches programming courses. In the second lecture, we allowed the subjects to use the learning environment to understand nested loops. In both experiments, we recorded the activities on the PC's display in order to observe

and analyze the subjects' interactions with our learning environment and to determine the subjects' understanding of programs that include nested loops. We used BB FlashBack Express 3 for recording the screen video of subjects' interactions with the environment.

4.2 Method of the Experiment

We conducted a pre-test at the end of the first lecture to judge the subjects' understanding level of nested loops before using the learning environment. Then, we asked the subjects to learn two programs using the learning environment. The program for the first exercise, Ex1, includes nested loops and displays a pyramid shape using the asterisk character. In learning the first program, the teacher described how to use the learning environment, and we helped the subjects to use the environment. The program for the second exercise, Ex2, also includes nested loops and stores a triangle shape using the asterisk character into two-dimensional array. In learning the second program, neither the teacher nor we provided help to the subject in understanding the programs. The subjects were given 20 minutes to learn the second program. Then, we conducted a post-test at the end of the self-study (second program) to judge each subject's level of understanding of nested loops after using this environment. Finally, the subjects filled out a questionnaire, which we used for a subjective assessment of the experiment.

The programs used in the pre-test and the post-test are different; however, the questions are almost the same. Questions 1-1 and 1-2 ask the roles of the first and the second inner loops, respectively. Question 1-3 ask the role of the outer loop. Questions 2-1 and 2-2 ask about the values of the control variables to judge the learner's skill in program tracing. Question 3 asks the purpose of the program as a whole.

4.3 Verifications of Hypotheses and Review of Results

To determine the effectiveness of our learning environment, we eliminated 6 subjects who had already fully understood nested loops in the pre-test. Then, we eliminated 6 other subjects who failed to record their own PC activities. Therefore, we analyzed 18 recordings.

For **Hypo1** and **Hypo2**, we checked whether the subjects performed the 14 actions when subjects were learning the two programs. Then there are the 2 actions for **Hypo2**. For each action, we categorized the 18 subjects into those who performed the action (positive group) and those who did not perform the action (negative group).

In some actions, the ratios of subjects in the positive group versus the negative group are more than 2 or less than 0.5. In those cases, smaller groups had less than 6 members, and the performance of an individual largely affected the performance of the whole group. Therefore, we eliminated those cases from the evaluation. Thus we selected the 6 actions that we could examine in detail, as follows:

- Act 1: The subject tries to pack the operation sequences into a package for the outer loop on Ex2
- Act 2: The subject tries to tag all packages on Ex2
- Act 3: The subject correctly tags all packages on Ex2
- Act 4: The subject tries to pack a nested package on Ex2
- Act 5: The subject correctly packs the nested package on Ex2
- Act 6:The subject fully confirms and observes all states of the target domain STD that corresponds to all program fragments P on Ex2

First, we analyzed the relationships between the pre-test score and these actions, and reaped the result of no significant difference between the negative and positive on an independent t-test.

We analyzed the elevated score from pre-test to post-test. As results, we reaped the following conclusions regarding the effectiveness of the actions (if there is a significant difference between the negative and positive with 5% significance level by performing an independent t-test, we put an asterisk after the act number. And, if there is a significant difference between the negative and positive with 1% significance level, we put two asterisks after the act number):

- Act 2^{*} is effective in understanding a single loop in Q1-1.
- Acts 1^{**}, 2^{*}, 4^{**} and 5^{*} are effective in understanding the outer loop in Q1-3.

- Acts 1^{**}, 2^{*}, 4^{**} and 5^{**} are effective in understanding the program as a whole in Q3.
- Act 6^{*} is effective in gaining the skill of program tracing in Q2-1.

On the other hand, for **Hypo3**, we analyzed the relationship between the post-test score and these actions. we reaped the following conclusions regarding the effectiveness of the actions:

- Acts 1^{**}, 2^{*}, 3^{*} and 4^{*} are effective in understanding a single loop in Q1-1.
- Acts 1^{**}, 4^{**} and 5^{*} are effective in understanding the inner loop in Q1-2.
- Acts 1^{*}, 2^{*}, 4^{**} and 5^{*} are effective in understanding the outer loop in Q1-3.
- Acts 1^{**}, 2^{*}, 4^{**} and 5^{*} are effective in understanding the program as a whole in Q3.

In all pairs of acts and questions (pairs of Acts 1, 2, 3, 4, 5 and Questions 1-1, 1-2, 1-3, 3, and pairs of Act 6 and Question 2-1, 2-2), the positive group achieved higher post-test scores than the negative group. In more than 68% of pairs, there are significant differences between the negative and positive with less than 5% significance level. It means that an understanding level of a member of positive groups tends to be better than of negative groups. Therefore, we suggest that teachers are able to know each learner's level of understanding by observing whether the learner performs actions of Act1 to Act6.

Finally, we summarized the questionnaire responses from 34 subjects (4 subjects did not join the experiment and only used our environment). The first question in the questionnaire asked whether the GUI interface is easy to view. The subject had four choices: "4 points: very easy to view", "3 points: easy to view", "2 points: difficult to view", "1 point: very difficult to view". 31 subjects gave the environment 3 or more points. The second question asked whether the GUI interface is easy to use and offered four similar choices. 29 subjects gave the environment 3 or more points. The third question asked whether they gained a deep understanding of nested loops. 26 subjects gave the environment 3 points. As those results, we suggest that the environment is useful for learning algorithm and programs.

5. Conclusion

We adopt the existing algorithm and programming learning environment in actual classroom. The environment allows the visualization of the relationships among the programs field (**PF**), the target domain field (**TDF**), and the operations field (**OF**). This paper attempts to support programming beginners in understanding program code that includes nested loops. We evaluated this environment in a classroom setting. The results suggest that our proposed environment raises the understanding level of programming beginners.

Acknowledgements

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

References

Bennedsen, J., & Caspersen, M. E. (2008). Optimists have more fun, but do they learn better? On the influence of emotional and social factors on learning introductory computer science. *Computer Science Education, 18*(1), 1-16.

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.

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.

Gabor, T., (2009). Algorithm visualization in programming education, . *Journal of Applied Multimedia*. *4*(3), 68-80.

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

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

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y. B. D., & Wilusz, T. (2001). A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *ACM SIGCSE Bulletin*, *33*(4), 125-180.

Noguchi, Y., Nakahara, T., Kogure, S., Konishi, T., & Itoh, Y. (2010). Construction of a learning environment for algorithm and programming where learners operate objects in a domain world', *International Journal of Knowledge and Web Intelligence*, *1*(3), 273-288.