Development of a Learning Support System for Source Code Reading Comprehension

Tatsuya Arai[1], Haruki Kanamori[1], Takahito Tomoto[2], Yusuke Kometani[1], and Takako Akakura[2]

[1] Graduate School of Engineering, Tokyo University of Science, 1-3 Kagurazaka, Shinjuku-ku, Tokyo 162-8601, Japan
{arai_tatsuya,kanamori_haruki,kometani}@ms.kagu.tus.ac.jp
[2] Faculty of Engineering, Tokyo University of Science, 1-3 Kagurazaka, Shinjuku-ku, Tokyo 162-8601, Japan
{tomoto,akakura}@ms.kagu.tus.ac.jp

Abstract. In this paper, we describe the development of a support system that facilitates the process of learning computer programming through the reading of computer program source code. Reading code consists of two steps: reading comprehension and meaning deduction. In this study, we developed a tool that supports the comprehension of a program's reading. The tool is equipped with an error visualization function that illustrates a learner's mistakes and makes them aware of their errors. We conducted experiments using the learning support tool and confirmed that the system is effective.

Keywords: Programming Learning, Problem Posing, Reading Program, ICT.

1 Introduction

This paper describes the development of a support system that facilitates the process of learning to write computer programs by reading computer program source code. In this study, we define reading source code as working backward from the code to determine the original requirement that led to the program. The process of reading code consists of two steps: reading comprehension and meaning deduction (see Fig. 1).

Information technology has spread throughout society, but there is a shortage of information engineers, and so it is necessary to train many more. There has been extensive research on how computer programming can be learned through the construction of computer programs [1]. However, obtaining deep understanding of programming requires learners to read source code in addition writing programs [2].

Programming experts are highly skilled at reading code because this skill is essential for debugging programs and inferring their purpose [3]. Reading code is an also important activity for gaining a deeper understanding of programming. Furthermore, posing problems is often useful in understanding the scope of a computer program [4]. Accordingly, we developed a support system that facilitates the process of learning to program by reading source code. The target learners are programming beginners.

S. Yamamoto (Ed.): HIMI 2014, Part II, LNCS 8522, pp. 12–19, 2014.

2 The Process of Programming

In previous research, the process of programming has been considered to consist of two steps: algorithm design and coding. Algorithm design is the step in which structures, such as flow diagrams, are used to construct the abstract process from the program's requirements. This processing flow is independent of the programming language. In contrast, coding is the step in which the abstract flow is converted into source code, which necessarily depends on the programming language. In learning to program, learners are often given problems in the form of requirements and asked to write the appropriate source code by first considering the abstract processing flow.

Our research group considers reading code to be an important skill that adds to the process of programming. We previously proposed that the process of reading code consists of two steps: reading comprehension and meaning deduction [6] (see Fig. 1). Reading comprehension is the inverse of coding, and meaning deduction is the inverse of algorithm design. In reading comprehension, learners are required to convert source code into an equivalent abstract processing flow. In meaning deduction, learners are required to deduce a requirement from the abstract processing flow. We developed a learning support system for meaning deduction [6]. However, reading comprehension was still not supported. Thus, in this study, we design a learning support system to foster reading comprehension, and we evaluate the effectiveness of the system.

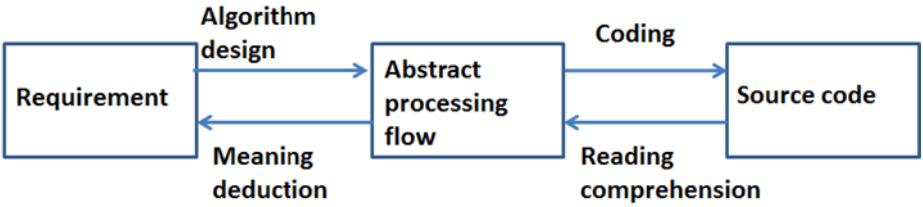

Fig. 1. The process of programming

3 Learning by Using a Flowchart

During the reading comprehension step, learners construct flowcharts from given pieces of source code. A flowchart has the advantage of making a problem (here, a requirement) more likely to be discovered by representing it visually. Figure 2 shows the process of constructing a flowchart. A learner chooses a series of flowchart blocks and populates each block with one of several available statements. Next, they populate each empty rectangular box with concepts. Finally, they connect the flowchart blocks with lines. By reducing the degrees of freedom in the answer, it is easier to convey the intent of the program.

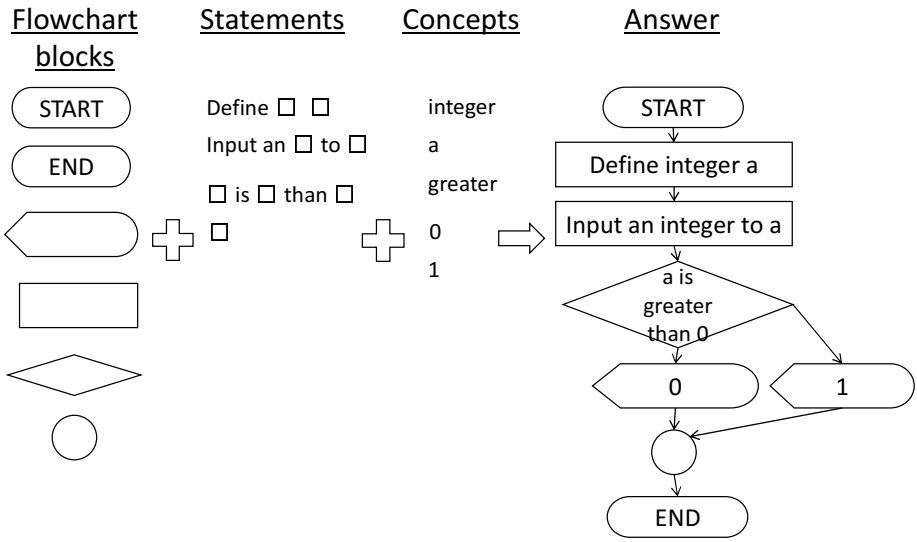

Fig. 2. How to write a flowchart

4 Error Visualization

Error visualization is the process of illustrating error [5]. The use of feedback allows teaching the correct answer and pointing out errors, but the learner stops thinking about the problem if simply shown the correct answer. If a learner is only shown their own mistakes, they are not able to understand how and why they erred. In contrast, illustrating errors can make the learner aware of their own errors. On this basis, we developed a learning support system that includes an error visualization function.

5 Experiments

We conducted two experiments with two different objectives: the objective of Experiment 1 was to examine the reading skill level of learners; the objective of Experiment 2 was to examine the influence of reducing the degrees of freedom of an answer.

5.1 Experiment 1

In Experiment 1, we spent 10 minutes explaining the principles of writing a flowchart to 62 second-year university students who were attending a programming course. The students were asked to solve 4 reading comprehension problems in 20 min, 4 algorithm design problems in another 20 min, and 4 coding problems in a final 20 min. Problems were given in a free-response format, and the maximum possible score for each problem was 2 points.

Table 1 shows the results of Experiment 1. The average score on individual problems was 1.20 for the reading comprehension exercise, 1.21 for the algorithm design exercise, and 1.69 for the coding exercise. From these results, we can conclude that

reading comprehension and algorithm design were difficult. Although algorithm design is often considered to be more difficult than coding, reading comprehension was found to be as difficult as algorithm design.

Table 1. Experiment 1 Results

	Average score per problem	Standard deviation
Algorithm design	1.21	0.52
Coding	1.69	0.39
Reading comprehension	1.20	0.38

5.2 Experiment 2

In Experiment 2, we explained the principles of writing a flowchart to 12 fourth-year university students for 10 min. After the explanation, the students were asked to solve 6 reading comprehension problems in 30 min, followed by 6 meaning deduction problems in 15 min. The types of answers permitted are shown in Sections 3 and 4. The maximum score for each problem was 2 points.

Table 2 shows the results of Experiment 2. The average score on individual problems was 1.21 for the reading comprehension exercise and 0.64 for the meaning deduction exercise. From these results, we can conclude that the effect of reducing the degrees of freedom of the answer was small, and that the meaning deduction exercise was a difficult task. From Experiments 1 and 2, we confirmed the need to develop a support system that facilitates the process of learning to program by reading code.

Table 2. Experiment 2 Results

	Average score per problem	Standard deviation
Reading comprehension	1.21	0.55
Meaning deduction	0.64	0.21

6 Learning Support System

6.1 Learning Screen

Figure 4 shows the learning screen of the learning support system. The learner uses flowchart blocks, statements, and concepts to construct a flowchart. First, a learner presses a flowchart block button, which makes that flowchart block appear in the center panel of screen. Next, the learner presses a statement button, which brings that statement (with blanks) to the answer column of the selected flowchart block. Next, the learner presses a concept button and selects a blank entry in a statement, which

inserts the selected concept into the selected blank space. When the learner has completed an answer, he or she presses the answer button. If the answer is correct, a message of "Correct answer" is displayed; if the answer is incorrect, the system shows the feedback screen (see next section).

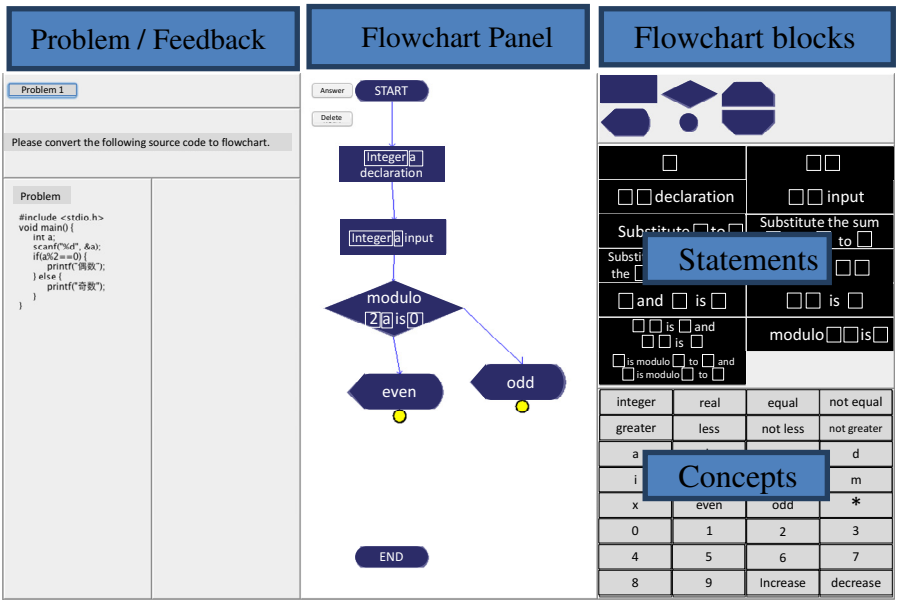

Fig. 3. Learning screen

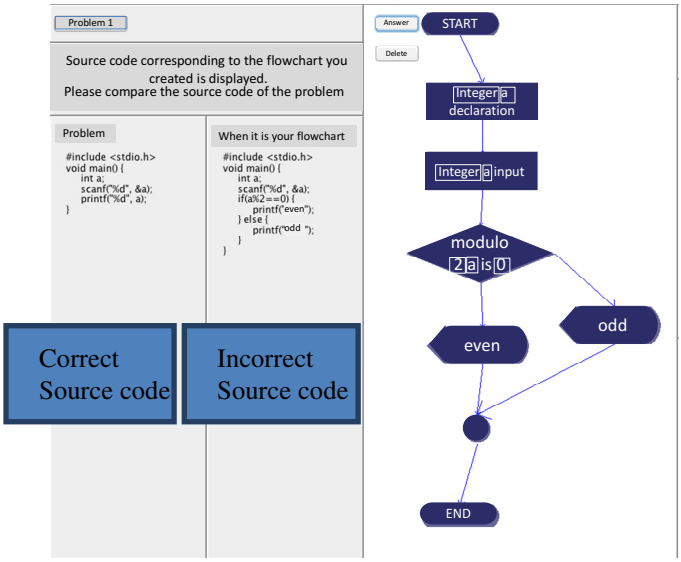

Fig. 4. Feedback screen

6.2 Feedback Screen

Figure 4 shows the feedback screen. If a learner reads the source code incorrectly, the system generates incorrect source code from the incorrect flowchart data, and the learner then looks for mistakes by comparing the incorrect source code to the correct source code.

7 Assessment Experiment

To ascertain the usefulness of the learning support system, we conducted an assessment experiment. In the assessment experiment, we administered a pre-test and a post-test to all participants (6 second-year university students). They are programming beginners. The pre-test and the post-test are identical. There are no feedback of test results to participants.

In the pre-test, after explaining the principles of writing a flowchart for 10 minutes, the participants were asked to solve 6 reading comprehension problems in 15 min. Next, the participants were divided into two groups: an experimental group (4 students) and control group (4 students). We spent 5 min explaining to the experimental group how to use the system.

The experimental group learned by solving 10 reading comprehension problems within 60 min using our system. In the control group, the participants were asked to solve the same problems as the experimental group by pen and paper in 60 min. The control group was allowed to view the correct answer. After this, participants of both groups were asked to take a post-test.

The 6 questions used in the pre-test and the post-test are as follows.

Q1: Flowchart with condition blocks. Basic if-else is included.

Q2: Flowchart with iteration blocks. A normal while loop is included.

Q3: Flowchart with condition blocks and iteration blocks. Both blocks from Q1 and Q2 are included.

Q4, Q5, Q6: Flowchart with nested structures combining condition blocks and iteration blocks.

Table 3 shows the coincidence between participant's answers and correct source codes. Although differences can be seen by comparing pre-test results and post-test results, there was no significant difference between control group and experimental group. Thus, we confirm the contents of the answers.

Table 3. Coincidence between participant's answers and correct answers

Group	participants	wholly coincident			partially coincident (%)		
		pre-test	post-test	pre-post difference	pre-test	post-test	pre-post difference
Control	A	3	4	1	0.56	0.80	0.24
	B	1	3	2	0.31	0.75	0.44
	C	0	0	0	0.26	0.58	0.32
Experimental	D	3	4	1	0.46	0.67	0.21
	E	0	3	3	0.38	0.72	0.34
	F	1	1	0	0.43	0.50	0.08

Subjects A, B (control group) and D (experimental group) gave correct answers with regard to the questions that they can answer within the time limit both in pre-test and in post-test, except for caress mistakes. Therefore, it is assumed that the improvements in test scores of A, B, D depended on their improvements in answer speed.

Subject C, in the pre-test, was not able to use the appropriate flowchart block for the while statement or if statement. In addition, C was not able to describe the appropriate sentence in the flowchart block. For example, when describing "Output an integer" or "Input an integer", C did not describe the variable name specifically. In addition, C was not able to describe the nested structure combining if statements and while statements. In post-test, C became possible to write more specifically the content of the block except for formulas. Furthermore, C was able to use the appropriate flowchart block with respect to the while statement. But still, C was not using the conditional branch block and didn't describe the structure of the conditional branch.

Subject E, in the pre-test, was not able to accurately describe the structure of if statements and while statements. For example, E wrote only one arrows from if block, and didn't write junction blocks. As for while statements, E described pre-determined condition as a post-judgment of iterations. In addition, E didn't describe formulas with natural languages. However, in the post-test, after conducting learning using the learning support system, E became able to write accurate structure of the conditional branch. In addition, E was aware of the structure of the while statement. Furthermore, E was able to explain properly the contents of the formula with natural languages.

In the pre-test, subject C and subject E was not able to adequately describe the flow chart of if and while statements. In the post-test, C was able to correct the error of the while statement for simple problems such as Q2. However, C did not adequately describe the while statement in a complex nested structures such as Q4. Furthermore C did not correct the error on the flow chart of if statements. In contrast with C, subject E who had used the system was accurately describe both if and while statements.

Table 4 shows the percentage of partially coincidence between correct source code and the source code corresponding to flowchart created by subjects. In Q4, the flowchart included both if statements and while statements and had nested structures. The score of Q4 by subject C decreased from 0.36 to 0.29, and the score of Q4 by subject E was improved from 0.64 to 0.86. This result shows the visualization of error using the learning support system are useful for learners in understanding to the flow of processing, such as conditional branches and iterations.

Table 4. Results of partially coincidence

Group	participants	pre-test							post-test						
		Q1	Q2	Q3	Q4	Q5	Q6	avg.	Q1	Q2	Q3	Q4	Q5	Q6	avg.
Control	A	1.00	1.00	1.00	0.36	0.00	0.00	0.56	1.00	1.00	1.00	1.00	0.78	0.00	0.80
	B	1.00	0.60	0.29	0.00	0.00	0.00	0.31	1.00	1.00	1.00	0.86	0.65	0.00	0.75
	C	0.38	0.40	0.29	0.36	0.09	0.04	0.26	0.88	0.90	0.57	0.29	0.78	0.08	0.58
Experimental	D	1.00	1.00	0.50	0.29	0.00	0.00	0.46	1.00	1.00	1.00	1.00	0.04	0.00	0.67
	E	0.38	0.50	0.50	0.64	0.26	0.00	0.38	1.00	1.00	1.00	0.86	0.43	0.00	0.72
	F	1.00	0.70	0.50	0.36	0.00	0.00	0.43	1.00	0.80	0.71	0.50	0.00	0.00	0.50

8 Conclusions and Future Work

In this study, we developed a learning support system to provide guidance on reading comprehension, and evaluated the effectiveness of our system. From the results of the assesment experiment, we confirmed that it is necessary to support learning of reading comprehension, and that our system is effective for doing so. However, the assessment experiment did not include many participants, and the number of participants should be increased in future experiments. Additionally, we did not develop a learning support system for guidance on both reading comprehension and the meaning deduction process, but we believe such a system should be developed in the future.

References

1. Matsuda, N., Kashihara, A., Fukukawa, K., Toyoda, J.: An instructional system for constructing algorithms in recursive programming. In: Proc. of the Sixth International Conference on Human-Computer Interaction, Tokyo, Japan, pp. 889–894 (1995)
2. Corbi, T.A.: Program understanding challenge for the 1990s. IBM Syst. J. 28(2), 294–306 (1989)
3. Uchida, S., Kudo, H., Monden, A.: An experiment and an Analysis of debugging process with periodic interviews. In: Proceedings of Software Symposium, Japanese, vol. 98, pp. 53–58 (1998)
4. Lyn, D.: Children's Problem Posing within Formal and Informal Contexts. Journal of Research in Mathematics Education 29(1), 83–106 (1998)
5. Hirashima, T.: Error-based simulation for error-visualization and its management. Int. J. of Artificial Intelligence in Education 9(1-2), 17–31 (1998)
6. Kanamori, H., Tomoto, T., Akakura, T.: Development of a Computer Programming Learning Support System Based on Reading Computer Program. In: Yamamoto, S. (ed.) HCI 2013, Part III. LNCS, vol. 8018, pp. 63–69. Springer, Heidelberg (2013)