東海大学紀要情報通信学部
Vol.9,No.1,2016,pp.8-14

Paper

Investigating the relationship between tracing skill and modification skill for different programming statements

by

Satoru KIKUCHI[*1] and Kazuhiko HAMAMOTO[*2]

(received on Mar.31, 2016 & accepted on May.26, 2016)

Abstract

In our study, we investigate the relationship between tracing and modification skills for different programming statements. The experiment comprised two tasks, a tracing task and modification task, and asked questions about basic programming statements (e.g., selection, iteration, and nested statements). The result indicates the following three points: (1) it is possible that the test subjects applied a different strategy for selection statement knowledge and iteration statement knowledge, (2) for iterative code, a code-modification task is easier than a code-tracing task, (3) it is possible that experience with frequently occurring code facilitates the learning process, even if the code is complex.

Keywords: *Programming education, Skill hierarchy, Tracing skill, Modification skill*

1. Introduction

Programming is an essential skill for computer science students, but it is not easy to improve a novice's programming skills. Many researchers all over the world have studied and proposed various kinds of teaching methods [1]-[4]. Those methods can be divided into two main categories according to the acquired knowledge. The methods of the first category emphasize comprehension of programming concepts and then expect the students to deduce from the knowledge. Conventional programming classes commonly follow this approach. The methods of the second category emphasize hands-on programming experience and then expect the students to apply the knowledge through induction. Both methods are used since the 1970s and are still used [5], [6].

Novice programmers, that have little experience such as a student who have taken only an introductory class at a university, can acquire an incorrect understanding of programming statements. This is called misconception and can even happen for simple statements [7]. Misconception causes inconsistent performance of students. For instance, it is possible that students can trace a simple code fragment but cannot trace slightly modified code, or they can trace code but cannot write similar code.

Study methods that emphasize an experience such as problem-based learning (PBL) [8] focus on acquiring a deeper understanding of a subject by solving problems but require teaching resources (i.e., tutors) and a comprehensive set of study questions. However, teaching resources are limited, and only a few good questions are available [8]-[10].

For most programming statements, it is assumed that gaining an experience is effective for deep understanding. However, if there are statements that are acquired a deeper understanding by conventional method, teachers can employ experience-based methods only for the experience-relevant statements. This allows them to create good questions effectively and saves teaching resources. However, it does not take into account that one requires experience for a deeper understanding of all programming statements. In this study, we investigate the relationship between program tracing and writing skills for all statements and examine statements that are relevant to know for a deeper understanding.

2. Related work

Previous research has investigated the relationship between different programming skills and suggested a skill hierarchy, with the skill of tracing iterative code and the skill of explaining code at a lower level than the skill of writing code [11]. This means that at least tracing and explanation skills are required to write code.

In [12], the authors also investigate a hierarchy of programming skills and added code-modification skill to the analysis. Their results do not completely match the results of [11]. The relationship between the skill of explaining code and the skill of writing code is the same as in [11]. However, tracing skill and writing skill are different and not directly correlated with each other. The skill of modifying non-iteration statements ranks lower than the skill of tracing non-iteration statements, and the skill of tracing iteration statements ranks lower than the skill of modifying iteration statements. It means the skill of modifying non-iteration statements is required to read code, and reading of iteration statements is required to modify iteration statements.

*1 Graduate School of Science and Technology, Doctoral Program, Tokai University
*2 Department of Information Media Technology, School of Information and Telecommunication Engineering, Professor, Tokai University

Table 1 Question list of the experiment. Both tracing and modification tasks use same question list.

Question name (question ID)	Description
Output 1 (Ot-1)	Output of variable
Selection 1 (Sl-1)	Selection + output
Selection 2 (Sl-2)	Selection + output
Iteration 1 (It-1)	Iteration + output. Afterthought is +1
Iteration 2 (It-2)	Iteration + output. Afterthought is +2
Iteration 3 (It-3)	Iteration + output. Afterthought is -1
Iteration 4 (It-4)	Iteration + output. Afterthought is -2
Nested 1 (Ns-1)	Nested iteration-selection + output
Nested 2 (Ns-2)	Nested iteration-selection + output
Nested 3 (Ns-3)	Nested iteration-iteration + output

3. Method

3.1 Purpose of the experiment

It is difficult to investigate the relationship between programming skills accurately, as the results reported in [11] and [12] slightly differ. One reason that makes it difficult is the wide range of conditional statements in programming languages. The authors of [11] and [12] investigated proceedings, functions, iterations, variables, and character data. They studied the statements by tracing, explaining, writing, and modifying them. A small number of conditional statements can result in a significant number of related questions. This makes it difficult to compare the details of each condition, as time in an experiment is limited.

In this study, we focus on evaluating reading and modification skills, and we only consider selection, iteration, and nested (e.g., nested iteration-selection) statements. The questions for the reading and modification tasks use similarly structured code to minimize the difference in difficulty level. We prefer free-response questions to multiple-choice questions, as we thought a multiple-choice test would give the test subjects clues on tracing and modifying the code.

The purpose of a programming study is to acquire code writing skills with a deeper understanding. However, for novice programmers, it is difficult to write code from scratch even if they can read code of the same difficulty level. Because we used free-response questions, a test subject might not be able to answer some of the questions. This makes it difficult to analyze the comprehension level of test subjects. For the experiment, we, therefore, used a modification task and not a writing task so that we can gather enough data for the analysis.

3.2 Detail of the experiment

The experiment comprised two tasks, one tracing task and one modification task. Both tasks had ten questions each and employed similarly structured code of introductory difficulty level. Table 1 lists the questions.

Table 2 Detail of point allocation for each question

Type	Score	Description
Tracing task	50%	Score of output number. (e.g. In case correct answer has 5 output values, subject get 50% points when the subject write 5 values, and get 40% points when 4 or 6 values are written.). The unit of counts are based on output functions usage, so a different format output is not counted as a correct answer.
	30%	Score of calculated values. (e.g. In case a correct answer has 5 output values, subject get 30% points if all written values are correct, and get 24% points when 4 values are correct.)
	20%	Score of output format. Subject get 20% points if a written answer is a correct format, and subtracted if the format is incorrect. (e.g. Get no score in case line break or space are used for separation of values)
Modification task	50%	Score of modification parts. (e.g. In case correct answer has 5 correction parts, get 50% points when all modification parts are modified, and get 30% points when only 3 parts are modified. Scores are subtracted in case wrong prats are modified. Subtracted 25% points if 5 wrong prats are modified)
	30%	Score of process result values. (e.g. In case a correct answer has 5 output values, subject get 30% points if all written values are correct, and get 24% points when 4 values are correct.)
	20%	Score of syntax. Subject get 20% points if a written code has no syntax error, and subtracted 20% points for errors (e.g. forget a semi-colon, forget variable declaration)
	Remarks	Subject gets no point when there is no modification to the code.

The first task was code tracing. Subjects were given code fragments written in the programming language C and asked to write the process result for each question. The second task was code modification. Subjects were given code fragments and the required result that differed from the process result of the given code. The subjects had to correct the code such that it produced the required result.

The questions were designed to investigate the depth of understanding of each C statement at an introductory level, so the code fragments gradually changed from easy (e.g., simple selection) to difficult (e.g., nested iteration). Question 1 (Ot-1) focusses on the output of code variables. Question 2 and Question 3 (Sl-1, Sl-2) ask for a simple code selection. Question 4 to Question 7 (It-1 to It-4) deal with simple iterative code fragments. Question 8 and Question 9 (Ns-1, Ns-2) look into nested iteration-selection

Table 3 Classification result of tracing and modification task.

Type	Tracing task	Modification task
High group	11	14
Low group	7	4

statements. Question 10 (Ns-3) is about a nested iteration-iteration statement. The appendix gives more details about the questions.

We analyzed the result by comparing the scores of the high tracing score group and the low tracing score group for each question of both tasks.

Before the experiment, the test subjects filled out a questionnaire and provided information about their grades for introductory and advanced programming classes and about their programming experience outside of class. The introductory class covers basic programming statements such as selection and iteration. The advanced class covers advanced elements such as file IO, functions, and structure. Both classes follow the conventional teaching method and are not experience-based.

4. Result

Eighteen subjects completed the experiment, and all of them had taken a programming class at University.

We allocated a score of 100 for the tracing and modification tasks and distributed the points equally over the questions. Table 2 provides details on point allocation for each question. Since the goal of the experiment is to investigate the depth of understanding of programming concepts, we consider comprehension of programming structure the most important aspect. Thus, we awarded half of the points for programming structure comprehension. In the tracing task, the most important aspect is the number of output values, and in the modification task, the most important aspect is the place of code modification.

The average score for the tracing task was 73.9 (SD=28.1), and the average score for the modification task was 76.6 (SD=24.7).

4.1 Classification of the subjects

We divided the test subjects into two groups according to two different aspects.

The first aspect is how well they scored in the tracing task. We used hierarchical cluster analysis with Ward's method to measure the proximity between groups of variables, and Squared Euclidean distance to measure the distance of the variables. This analysis put eleven subjects into the high tracing group and seven subjects into the low tracing group. The second aspect is how well they scored in the modification task. The analysis put fourteen subjects into the high modification group and four subjects into the low modification group. Table 3 shows the result.

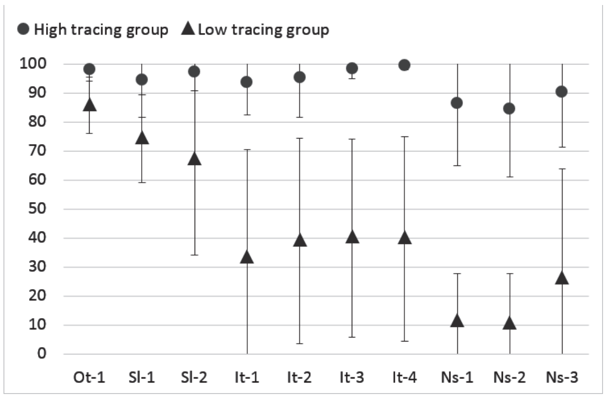

Fig.1 Average score of the tracing task for high and low tracing group. X-axis is question ID, Y-axis is score of the task and error bar shows SD.

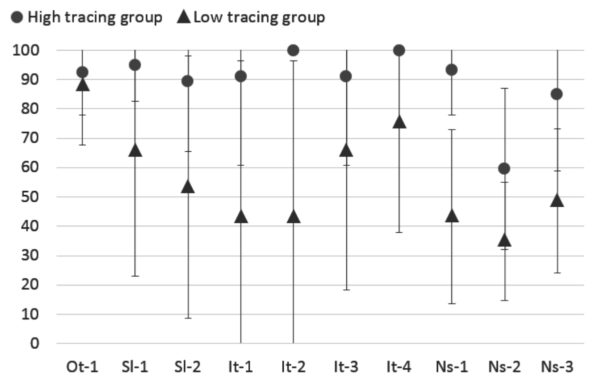

Fig.2 Average score of the modification task for high and low tracing group. X-axis is question ID, Y-axis is score of the task and error bar shows SD.

4.2 Score comparison between high and low tracing groups

Fig.1 and Fig.2 show the average scores of the high tracing group and the low tracing group. Table 4 compares the tracing scores of the high and low tracing groups, while Table 5 compares their modification scores. We employed one-tailed Welch's t test for the comparisons and Bonferroni modification in case of multiple comparisons.

In the tracing task, four questions about iteration (It-1 to It-4) revealed a significant difference at a level of p = 0.05 between the high tracing group and the low tracing group. Two questions about iteration with selection (Ns-1, Ns-2) revealed a significant difference at a level of p = 0.001 and question about nested iteration-iteration (Ns-3) revealed a significant difference at a level of p = 0.05. Other questions did not reveal a significant difference.

In the modification task, only one question about nested iteration-selection (Ns-1) revealed a significant difference at a level of p = 0.05. No other questions revealed a significant difference.

4.3 Comparison of questionnaire answers between high and low groups for each task.

Table 6 compares the programming experience between

Table 4 Score comparison of tracing task between high group and low tracing group. (Bonferroni corrected p-values)

ID	Group	Mead (SD)	Difference
Ot-1	High	98.2 (4.0)	
	Low	85.7 (9.8)	t(7.3)=3.21,p=0.140
Sl-1	High	94.5 (12.9)	
	Low	74.3 (15.1)	t(11.4)=2.93,p=0.133
Sl-2	High	97.3 (6.5)	
	Low	67.1 (33.0)	t(6.3)=2.39,p=0.525
It-1	High	93.7 (11.3)	*
	Low	33.1 (37.4)	t(6.7)=4.16,p=0.046
It-2	High	95.4 (13.8)	*
	Low	39.1 (35.4)	t(7.2)=4.01,p=0.049
It-3	High	98.5 (3.5)	*
	Low	40.0 (34.2)	t(6.1)=4.52,p=0.039
It-4	High	99.5 (1.5)	*
	Low	39.7 (35.3)	t(6.0)=4.48,p=0.042
Ns-1	High	86.5 (21.5)	***
	Low	11.1 (16.5)	t(15.2)=8.38,p<0.001
Ns-2	High	84.5 (23.5)	***
	Low	10.3 (17.3)	t(15.5)=7.70,p<0.001
Ns-3	High	90.3 (19.1)	*
	Low	25.9 (38.0)	t(8.0)=4.16,p=0.032

Table 5 Comparison of modification task between high group and low tracing group. (Bonferroni corrected p-values)

ID	Group	Mead (SD)	Difference
Ot-1	High	92.3 (14.4)	
	Low	87.9 (20.2)	t(9.9)=0.50,p=1.0
Sl-1	High	95.0 (12.4)	
	Low	65.7 (42.8)	t(6.7)=1.77,p=1.0
Sl-2	High	89.5 (24.1)	
	Low	53.3 (44.7)	t(8.3)=1.97,p=0.833
It-1	High	90.9 (30.2)	
	Low	42.9 (53.5)	t(8.5)=2.17,p=0.601
It-2	High	100 (0.0)	
	Low	42.9 (53.5)	t(6.0)=2.83,p=0.300
It-3	High	90.9 (30.2)	
	Low	65.6 (47.2)	t(9.1)=1.26,p=1.0
It-4	High	100 (0.0)	
	Low	75.4 (37.5)	t(6.0)=1.73,p=1.0
Ns-1	High	93.2 (15.4)	*
	Low	43.3 (29.6)	t(8.1)=4.12,p=0.033
Ns-2	High	59.6 (27.5)	
	Low	34.9 (20.1)	t(15.5)=2.20,p=0.430
Ns-3	High	84.9 (26.1)	
	Low	48.6 (24.5)	t(13.5)=2.99,p=0.101

Table 6 Summary count of programming experience questionnaire answer of high and low group in each task. Value in a cell is a number of subjects.

	Group	Tracing task	Modification task
Yes	High	7	7
	Low	0	0
No	High	4	7
	Low	7	4

Table 7 Summary count of introductory programming class grades questionnaire answer for high and low group of each task.

Introductory	Group	Tracing task	Modification task
S	High	2	3
	Low	1	0
A	High	3	4
	Low	2	1
B	High	1	1
	Low	0	0
C	High	4	5
	Low	4	3
N	High	1	1
	Low	0	0

Table 8 Summary count of advanced programming class grades questionnaire answer for high and low group of each task.

Advanced	Group	Tracing task	Modification task
S	High	3	3
	Low	0	0
A	High	4	4
	Low	0	0
B	High	0	0
	Low	0	0
C	High	0	1
	Low	2	1
N	High	4	6
	Low	5	3

the high and low tracing groups and the high and low modification groups. The question asked in the questionnaire was “Do you often program in your private time?” All seven subjects who answered “YES” belonged to the high tracing group and the high modification group. The other subjects who answered “NO” belonged to high and low groups. With respect to tracing skills, four subjects were ranked as high and seven were ranked as low. With respect to modification skills, seven were ranked as high and four were ranked as low. Three subjects who answered “NO” were classified into the low group for tracing but into the high group for modification.

Table 7 shows the subjects’ grades for introductory programming summarized for the high and low tracing and modification groups. Table 8 shows the same analysis for the advanced class. Grade S is the best grade, and Grade C is the lowest grade. Grade N means that the subject failed to earn a credit or that s/he did not take the class. For the grades in introductory programming, the classification result differs between the tasks for the grades S, A, and C. Three subjects were classified into the low tracing group but classified into the high modification group. For the grades in advanced programming, the classification result differs for the grades C and N.

5. Discussion

The results for selection and iteration show a different tendency. Comparing the scores between the high and low tracing groups for answering questions on iteration statements (It-1 to It-4), we can only see a significant difference for the tracing task, not for the modification task. For questions on selection statements (Sl-1, Sl-2) the answers did not show a significant difference for both tracing and modification tasks. This result suggests the possibility that the subjects applied different solution strategies for the different statements. Knowledge about selection statements that the subjects acquired in a conventional method class tends to be suitable for deduction. The results for the tracing task do not differ between the high and low groups. This means that even the low group was able to read the code. However, knowledge about iteration statements does not seem well suited for deduction. The tracing score is different between the high and low groups, and there is a possibility that the subjects applied other strategies such as induction or analogy. The modification score of the low group is not worse than their tracing score, and the score is the same as the score of the high group. This means that subjects from the low group could not read iteration code correctly, but they somehow found the mistakes in the code for the modification task. The knowledge that cannot read but can modify was acquired from a conventional teaching method that does not focus on gaining much experience with writing and modifying iteration statements, so the subjects developed a misconception of iteration statements. Considering an experience-based method such as PBL that focuses on acquiring a thorough understanding, the assumption is that we can avoid misconceptions by gaining experience in writing or modifying iteration statements.

The results for nested iteration-selection (Ns-1, Ns-2) were different for each question. For the analysis of the knowledge about nested iteration-selection, Ns-1 result is preferred. Because, Ns-2 is a question that contains not only about simple programming statement but it also requires additional knowledge as it uses remainder. Since Ns-1 shows a difference in both tasks, we can conclude that it is difficult to trace and modify nested iteration-selection statements for the low group. The reason for this could be a lack of tracing skill for iteration statements. However, the question about nested iteration-iteration (Ns-3) statements shows no difference in modification. The result indicates that nested iteration-iteration statements might be easier to understand than nested iteration-selection statements. This could be the case because nested iteration-iteration statements (e.g., initialization of a two-dimensional array, sort algorithm) do frequently occur and are considered standard code. However, this requires further research, and we need to study the comprehension of standard code fragments with increased sample size and code variations.

In [4], the authors suggest that a poor writing skill can be improved by progress in tracing, and they recommend a tracing study. From the result of the experiment, we can assume that performing a modification task before tracing could be effective as modification is easier than tracing for iteration statements. Students would then feel more comfortable to answer the questions, and that would lead to a reduced dropout rate for the class. For the study of complex code such as nested code, experience with similar code seems to be effective and creates background knowledge about standard code.

Our questionnaire showed that students with a misconception of iteration knowledge that were classified into the low group for tracing but the high group for modification, can achieve good grades in the introductory class but got low grades or failed in the advanced class.

6. Conclusion

In this study, we investigated the relationship between tracing and modification skills for different programming statements. The result shows following three points: (1) it is possible that the test subjects applied a different strategy for selection statement knowledge and iteration statement knowledge, (2) for iterative code, a code-modification task is easier than a code-tracing task, and (3) it is possible that experience with frequently occurring code facilitates the learning process, even if the code is complex.

The results indicate that a teacher might customize his/her teaching method for different kinds of statements, that selection is best taught by a conventional method and iteration by an experience-based method, and that complex code is best taught by gaining experience with some standard code samples.

Future work will look into the following two points: (1) investigate whether experience with modifying iteration code affects the tracing iteration skill, and (2) inquire into the characteristic of frequently occurring code and whether it shows a similar result as nested iteration-iteration in this study.

References

[1] Pears, a et al., "A survey of literature on the teaching of introductory programming", SIGCSE Bulletin, Vol. 39. Issue 4, pp.204–223. (2007)

[2] Salleh, S. M et al., "Analysis of Research in Programming Teaching Tools: An Initial Review", Procedia - Social and Behavioral Sciences, Volume 103, pp.127–135 (2013)

[3] Sheard, J et al., "Analysis of research into the teaching and learning of programming", Proceedings of the Fifth International Workshop on Computing Education Research Workshop - ICER '09, 93. (2009)

[4] Lister, R et al., "A multi-national study of reading and tracing skills in novice programmers", ACM SIGCSE Bulletin. Vol. 36. (2004)

[5] Rolandsson, L. "Changing computer programming education: The dinosaur that survived in school: An explorative study about educational issues based on teachers' beliefs and curriculum development in secondary school", Proceedings - 2013 Learning and Teaching in Computing and Engineering, LaTiCE 2013, pp.220–223 (2013)

[6] Prince, M. J., Felder, R. M. "Inductive Teaching and Learning Methods: Definitions, Comparisons, and Research Bases", Journal of Engineering Education, Vol. 95, Issue 2, pp.123–138 (2006)

[7] Kaczmarczyk, L. C et al., "Identifying student misconceptions of programming", Proceedings of the 41st ACM Technical Symposium on Computer Science Education - SIGCSE '10, pp.107–111(2010)

[8] O'Kelly, J., Gibson, J. P. "RoboCode & problem-based learning", Proceedings of the 11th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education - ITICSE '06, Volume 38, Issue 3, pp.218-221 (2006)

[9] Nuutila, E et al., "Learning programming with the PBL method - Experiences on PBL cases and tutoring" Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 4821 LNCS, pp.47–67 (2008)

[10] Kinnunen, P et al., "Problems in Problem-Based Learning–Experiences, Analysis and Lessons Learned on an Introductory Programming Course", Informatics in Education-An International Journal, NO.4, Vol.2, pp.193–214(2005)

[11] Lopez, M et al., "Relationships between reading, tracing and writing skills in introductory programming", Proceeding of the Fourth International Workshop on Computing Education Research - ICER '08, pp.101–112. (2008)

[12] Mitsuo Y et al., "Research on programming skill hierarchy", IPSJ SIG Technical Report, Vol.2010-CE-104, No.3, pp.1-25 (2010)

Appendix: Question of each tasks.

Questions of tracing task. Program lines of define, main function and return is abbreviated.

Output 1 (Ot-1).

```
int a=20, b=40, c=10;
a = 10;
b = 15 + b;
c = 10 + a;
printf("%d,%d,%d", a, b, c);
```

Selection 1 (Sl-1).

```
int a = 20;
if(a > 30){
    printf("%d,", a);
    a = a + 40;
} else {
    printf("%d,", a);
    a = a + 10;
}
printf("%d", a);
```

Selection 2 (Sl-2).

```
int a = 20;
if(a > 35){
    printf("%d,", a);
    a = a + 20;
} else {
    printf("%d,", a);
    a = a + 30;
}
printf("%d", a);
```

Iteration 1 (It-1).

```
int a = 0, i;
for(i=2;i<=6;i++){
    a = a + i;
    printf("%d-%d,",i, a);
}
printf("%d",a);
```

Iteration 2 (It-2).

```
int a = 0, i;
for(i=2;i<=6;i=i+2){
    a = a + i;
    printf("%d-%d,",i, a);
}
printf("%d",a);
```

Iteration 3 (It-3).

```
int a = 0, i;
for(i=6;i>=2;i=i-1){
    a = a + i;
    printf("%d-%d,",i, a);
}
printf("%d",a);
```

Iteration 4 (It-4).

```
int a = 0, i;
for(i=6;i>=2;i=i-2){
    a = a + i;
    printf("%d-%d,",i, a);
}
printf("%d",a);
```

Nested 1 (Ns-1).

```
int a = 0, i;
for(i=1;i<=5;i++){
    if(i>=3){
        a = a + i;
    }
    printf("%d-%d,",i, a);
}
printf("%d",a);
```

Nested 2 (Ns-2).
```
int a = 0, i;
for(i=0;i<=5;i++){
    if(i%2==0){
        a = a + 2;
    }
    printf("%d-%d,",i, a);
}
printf("%d",a);
```
Nested 3 (Ns-3).
```
int a = 0, i, j;
for(i=0;i<=1;i++){
    for(j=0;j<=2;j++){
        a = a + 1;
        printf("%d-%d-%d,",i, j, a);
    }
}
printf("%d",a);
```

Questions of modification task. Program lines of define, main function and return is abbreviated.

Output 1 (Ot-1).
Preferred result: 20,50,55
```
int a=30, b=10, c=30;
a = 20;
b = 30 + a;
c = 25 + c;
printf("%d,%d,%d", a, b, c);
```
Selection 1 (Sl-1).
Preferred result: 30,80
```
int a = 30;
if(a > 10){
    printf("%d,", a);
    a = a + 50;
} else {
    printf("%d,", a);
    a = a + 20;
}
printf("%d", a);
```
Selection 2 (Sl-2).
Preferred result: 15,55
```
int a = 15;
if(a > 20){
    printf("%d,", a);
    a = a + 10;
} else {
    printf("%d,", a);
    a = a + 40;
}
printf("%d", a);
```
Iteration 1 (It-1).
Preferred result: 1-1,2-3,3-6,4-10,5-15,6-21,21
```
int a = 0, i;
for(i=1;i<=6;i++){
    a = a + i;
    printf("%d-%d,",i, a);
}
printf("%d",a);
```
Iteration 2 (It-2).
Preferred result: 1-1,3-4,5-9,9
```
int a = 0, i;
for(i=1;i<=6;i=i+2){
    a = a + i;
    printf("%d-%d,",i, a);
}
printf("%d",a);
```
Iteration 3 (It-3).
Preferred result: 6-6,5-11,4-15,3-18,2-20,1-21,21
```
int a = 0, i;
for(i=6;i>=1;i=i-1){
    a = a + i;
    printf("%d-%d,",i, a);
}
printf("%d",a);
```
Iteration 4 (It-4).
Preferred result: 6-6,4-10,2-12,12
```
int a = 0, i;
for(i=6;i>=1;i=i-2){
    a = a + i;
    printf("%d-%d,",i, a);
}
printf("%d",a);
```
Nested 1 (Ns-1).
Preferred result: 1-0,2-2,3-5,4-9,5-14,6-20,20
```
int a = 0, i;
for(i=1;i<=6;i++){
    if(i>=2){
        a = a + i;
    }
    printf("%d-%d,",i, a);
}
printf("%d",a);
```
Nested 2 (Ns-2).
Preferred result: 0-2,1-2,2-2,3-4,4-4,5-4,6-6,6
```
int a = 0, i;
for(i=0;i<=6;i++){
    if(i%3==0){
        a = a + 2;
    }
    printf("%d-%d,",i, a);
}
printf("%d",a);
```
Nested 3 (Ns-3).
Preferred result: 0-0-1,0-1-2,1-0-3,1-1-4,2-0-5,2-1-6,6
```
int a = 0, i, j;
for(i=0;i<=2;i++){
    for(j=0;j<=1;j++){
        a = a + 1;
        printf("%d-%d-%d,",i, j, a);
    }
}
printf("%d",a);
```