東海大学大学院平成30年度博士論文

# 経験により得られる能力に着目したプログラミング学習方法の提案

指導　　濱本　和彦 教授

東海大学大学院総合理工学研究科
総合理工学専攻

菊池　智

# abstract

This study proposes a learning method, which is focused on the abilities obtained from experiences, and it is evaluated by comparing to a conventional method. This approach is different from the conventional method that places emphasis on knowledge. I also analyze the programming abilities obtained by experience, and apply the findings to a proposing learning method. Programming education is a fundamental and important subject in a university information engineering curriculum. However, its learning is not easy, and even if a university student takes a lecture, it may not be possible to understand the basic contents. Thus, effective learning methods are required, and research is conducted all over the world. Research on programming education includes a wide variety of research subjects, including programming languages, teaching materials, models of understanding, and instruction methods. Although those approaches that focus on how to make most of an understanding and how to apply the understood knowledge as a method proposed so far, there is no research that is decisively effective as an educational effect. Meanwhile, as a survey of programming ability, there are reports that emphasize experience such as past programming experience has a great influence, reports that learning time is important rather than learning method. Also, there are some researches that emphasize an experience (learning by doing), and there are studies of positions thinking that experience is important. However, such methods have few theoretical backgrounds and there is no consistent theory and method. Therefore, there is a problem that the method is not established. In this research, I propose a consistent theory and method of experience–based learning method through theoretical investigation and practical investigation. Chapter 1 summarizes the above background and the purpose of this research. Chapter 2 introduces the problems of programming learning and various proposed methods. In Chapter 3, I summarize the findings about the ability gained through experience as cognitive scientific knowledge, redundancy of human cognitive ability, multilayered intelligence

and the theory of parallel processing. In Chapter 4, I propose a hypothesis of a learning method based on the findings. In chapter 5, I describe a survey experiment on the relationship between experience and programming ability. As an experiment, I investigated the influence of experience on programming ability. As a result, in the case of algorithm understanding, there was no score difference between the groups by experience ($p = 0.16$), but when implementing it, the difference in score was significant among the groups due to programming experience ($p<0.001$). Also, the reliability of inter-group differences of experience classification was higher than the inter-group score difference of the other classifications of introductory class ($p = 0.11$) and applied class results ($p = 0.016$). As a next experiment, I investigated the knowledge structure of programming. I asked 45 problems describing the code output results(Trace task) and classified those problems by cluster analysis. The parameters used for the analysis were the average point and variance of the problem and correlation coefficient between the score of trace task and score of writing task. As a result, typical processing (outputting values of control variables by adding values one by one) of an iterative statement was classified as easy problem, iteration statement with only character output was classified as medium level even if the same iteration control, the problem that the control variable was added more than two was classified as high difficulty. If there is knowledge of iterative processing defined like a textbook, it is considered that the above result will not be obtained. In the sense that part of elements are replaced, development of knowledge form as if a variable is shown, it can be considered as schema which is a representation of cognitive structure is related to programming knowledge. A schema is a specific knowledge structure (hierarchical structure, knowledge structure which can be replaced with a slot, default value at the time of information loss) obtained by a lot of experience. As another investigation, I conducted an analogy investigation on the ability of the approach other than deduction in programming. Comparing the score differences between the upper ranking point group and the lower ranking point group of the programming reading problem, seven out of ten questions differed in confidence intervals of 95% or more, but the result of programming code modify problem showed only one out of ten questions differ among the groups. Although the lower ranking point group does not understand the output result, they showed that it is partially possible to modify the code. The result showed the possibility that not only deductive method is used but also the analogical method is used together. Chapter 6 describes a survey on the impact of empirical learning methods on programming skills. In order to investigate the elements of programming (such as the functions of various functions),

I compared the scores of the post test after separating into groups that learn code by reading and learn code by writing. Since the writing learning group was not significantly different (p = 0.07) as a result of the output result description (trace) task, only the reading learning group was significantly different (p = 0.045). Therefore, if the same condition in element learning, reading learning could be more effective than writing learning. Another experiment investigating the learning of logic, which is a combination of multiple elements of programming. It compared the group (experience group) that learns by solving a large number of output results multiple choice problems and the conventional method (understanding with visualized teaching materials) group (understanding group). The result, which compared with the learning groups, was different depending on the question type of the posttest. In the experience group, the tracing problem improved (p = 0.003) but the understanding group did not improve (p = 0.069). However, in the task of memorizing the code, the opposite result was obtained, and the experience group (p = 0.524) understanding group (p<0.001) was found. However, the experience group (p = 0.027) was better than the understanding group (p = 0.156) in the reordering task of the code, and a certain effect was recognized. In chapter 7, as an optimal learning method derived from previous experiments, we propose a method that improves it based on a large number of output results multiple choice problems at the time of logic learning. The proposed method is an iterative questioning method of the same problem with the aim of efficiently acquiring the schema structure. For the purpose of forming slots and default values, we set up a main problem and iteratively answer it (Mail question → Similar question1 → Main question → Similar question2 → Main question → Similar question3 ...). In the experiment compared with the conventional learning method, in the writing task, since the proposed method (p = 0.004) contributed to the improvement of the pre/post score compared with the conventional method (p = 0.023), the effect equal to or better than the conventional method. In chapter 8, I summarize the above research results and make a conclusion. It was shown that the learning method, which answers many selection problems of output result proposed in this research was more effective in the writing and reading task in particular than the conventional method. It was shown that the other tasks were equivalent. A method of setting a main problem and iterating it is expected to be particularly effective. This method is expected to be useful because it can be applied to basic learning while paying attention to the ability obtained from experience, and because it has advantages that it is easy to apply as self learning and tasks.

# 目次

# 図目次

# 表目次

# 第 1 章

# 序論

## 1.1　背景

プログラミングは情報工学を学ぶ学生にとり重要な科目であるが，大学生においてもプログラミングの教育は容易ではなく，専門の講義を受講した後においても，十分な理解ができないことが報告されている [1][2].

教育の難しい点をまとめると次の４点が挙げられる.

**要素会得**　基本的な概念も理解が容易ではない [3]

**理解多様性**　多様な誤概念 [3]．結果，出来る者とそうでない者の差が広がる [4].

**動機づけ**　研究対象として脱落者というテーマが挙がるほど継続が容易ではない [5].

**知識応用**　コードを読むことが出来ても書けない [6]

プログラミング教育については世界中の研究者が問題点や改善方法について研究を行っている [1][7][6]．様々な提案があり，効果も一定は認められるが，これまでに突出して効果的であるという手法については報告されていない [8].

学習方針の大きな分類として，文献 [9] のように「演繹」と「帰納」という分類で分ける研究があり，また他の分類としても「理解」「経験」という分類をする研究もある [10]．文献 [10] によると，このような対立は 1970 年代より見られ，現状でもそれぞれの立場に支持があるものの「理解」に対する支持が多い．しかしながら，一定期間双方の立場が存在することは，それぞれの方針がともに一定以上有効であると見なすことができる.

理解を中心とした学習方法としては，基本的には知識の理解とその応用を重要視している [11]．したがって，根本的なプログラミング能力としては，要素会得とその演繹が前提となっ

ているとみなせる．

一方で，問題を解きながら理解をする手法など経験を重要視する学習方法もある [12]．学習においては学習方法ではなくて経験時間が重要であるという指摘や [13][14]，プログラミング言語を学ぶ事において別のプログラミング言語経験が大きな影響を与えるという報告がある [15] 通り，経験に着目した学習方法は効果が期待されている．反復的な学習としてドリル形式のような多数の問題に回答する手法も研究の対象となるが [16][17]，学習形式や出題内容について理論的な根拠を持つ提案にはなっておらず，そのような手法を適用する場合に最適な適用が行われているか判断することが難しい．

以上を踏まえると経験に関わる能力も学習において重要とみなせるが，理解型学習方法に常に取り込まれているわけではなく，必須の要件とみなされてはいない．また，経験型学習方法について，理論から手法まで一貫した研究がなされておらず，経験によって何が得られるかという理論的な根拠が明確になっていない．そして，どのような経験をした方が好ましいか議論されないため，学習形式や出題内容について最適な方法が明らかになっていないという課題がある．

## 1.2 本研究の目的

本研究ではプログラミング教育において，経験により得られる能力について，理論的な根拠を持ち，かつ，それに対応した学習方法の提案を行うとともに，提案した学習方法の検証を行う事で，経験に着目した一貫性のある学習の理論と方法の提案を行う．

また，本研究の対象としては主に大学生とし，入門レベルのプログラミング読みと書きスキルの向上とする．

## 1.3 本論文の構成

2 章ではプログラミング教育における課題を示す．2 章 1 節では，プログラミング学習の容易ではない点，学習の課題について記載する．これについては，大きく 4 点の課題として分類して記載する．2 章 2 節では，これまでに提案されている学習方法について「理解」「経験」という観点から，それぞれの分類における代表的な研究を紹介し，同時に課題も記載する．3 章では経験により得られる能力，推論機構，並列的な能力の使用について従来研究の知見を記載する. 4 章では経験により得られる能力に着目したプログラミング学習方法の仮説を提案する．5 章では理論的な調査研究として，プログラミング力と経験の関係，スキーマ構造，演繹以外

の推論の調査とその結果を記載する．6 章では実践的な調査研究として，提案する学習方法で効果的な要素の検証を従来手法との比較実験を通して行う．7 章では全ての実験の結果を踏まえ，その結果から導かれる学習方法の提案とその評価実験を記載する．8 章では，得られた知見をまとめ，課題と展望を記載する．

# 参考文献

[1] Michael McCracken, Vicki Almstrum, Danny Diaz, Mark Guzdial, Dianne Hagan, Yifat Ben-David Kolikant, Cary Laxer, Lynda Thomas, Ian Utting, and Tadeusz Wilusz. A multi-national, multi-institutional study of assessment of programming skills of first-year cs students. In Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education, ITiCSE-WGR '01, pages 125–180, New York, NY, USA, 2001. ACM.

[2] Lisa C. Kaczmarczyk, Elizabeth R. Petrick, J. Philip East, and Geoffrey L. Herman. Identifying student misconceptions of programming. In Proceedings of the 41st ACM Technical Symposium on Computer Science Education, SIGCSE '10, pages 107–111, New York, NY, USA, 2010. ACM.

[3] Takayuki Sekiya and Kazunori Yamaguchi. Tracing quiz set to identify novices' programming misconceptions. In Proceedings of the 13th Koli Calling International Conference on Computing Education Research, Koli Calling '13, pages 87–95, New York, NY, USA, 2013. ACM.

[4] Theresa Beaubouef and John Mason. Why the high attrition rate for computer science students: Some thoughts and observations. SIGCSE Bull., 37(2):103–106, June 2005.

[5] Syahanim Mohd Salleh, Zarina Shukur, and Hairulliza Mohamad Judi. Analysis of research in programming teaching tools: An initial review. Procedia - Social and Behavioral Sciences, 103:127 – 135, 2013. 13th International Educational Technology Conference.

[6] Raymond Lister, Elizabeth S. Adams, Sue Fitzgerald, William Fone, John Hamer, Morten Lindholm, Robert McCartney, Jan Erik Moström, Kate Sanders, Otto Seppälä, Beth Simon, and Lynda Thomas. A multi-national study of reading and tracing skills in novice programmers. SIGCSE Bull., 36(4):119–150, June 2004.

[7] Amjad Altadmri and Neil C.C. Brown. 37 million compilations: Investigating novice programming mistakes in large-scale student data. In Proceedings of the 46th ACM Technical

Symposium on Computer Science Education, SIGCSE '15, pages 522–527, New York, NY, USA, 2015. ACM.

[8] Arto Vihavainen, Jonne Airaksinen, and Christopher Watson. A systematic review of approaches for teaching introductory programming and their influence on success. In Proceedings of the Tenth Annual Conference on International Computing Education Research, ICER '14, pages 19–26, New York, NY, USA, 2014. ACM.

[9] Michael J. Prince and Richard M. Felder. Inductive teaching and learning methods: Definitions, comparisons, and research bases. Journal of Engineering Education, 95(2):123–138, 2006.

[10] L. Rolandsson. 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. In 2013 Learning and Teaching in Computing and Engineering, pages 220–223, March 2013.

[11] Judy Sheard, S. Simon, Margaret Hamilton, and Jan Lönnberg. Analysis of research into the teaching and learning of programming. In Proceedings of the Fifth International Workshop on Computing Education Research Workshop, ICER '09, pages 93–104, New York, NY, USA, 2009. ACM.

[12] Arto Vihavainen, Matti Paksula, and Matti Luukkainen. Extreme apprenticeship method in teaching programming for beginners. In Proceedings of the 42Nd ACM Technical Symposium on Computer Science Education, SIGCSE '11, pages 93–98, New York, NY, USA, 2011. ACM.

[13] Amir Kirsh and Iris Gaber. Satisfaction, time investment and success in students' programming exercise. In Proceedings of the Programming Experience 2016 (PX/16) Workshop, PX/16, pages 9–20, New York, NY, USA, 2016. ACM.

[14] G. Silva-Maceda, P. David Arjona-Villicaa, and F. Edgar Castillo-Barrera. More time or better tools? a large-scale retrospective comparison of pedagogical approaches to teach programming. IEEE Transactions on Education, 59(4):274–281, Nov 2016.

[15] Dianne Hagan and Selby Markham. Does it help to have some programming experience before beginning a computing degree program? SIGCSE Bull., 32(3):25–28, July 2000.

[16] Shuktika Jain. Automated generation of programming language quizzes. In Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015,

pages 1051–1053, New York, NY, USA, 2015. ACM.

[17] N. Funabiki, Y. Korenaga, Y. Matsushima, T. Nakanishi, and K. Watanabe. An online fill-in-the-blank problem function for learning reserved words in java programming education. In 2012 26th International Conference on Advanced Information Networking and Applications Workshops, pages 375–380, March 2012.

# 第 2 章

# プログラミング教育の課題と従来研究

## 2.1　プログラミング教育の課題

### 2.1.1　要素会得の課題

プログラミングにおいて最も代表的な課題は，プログラミングの各要素自体を理解する，要素を会得すること自体が難しい点である．特に，この課題はプログラミング言語の文法自体であったり，プログラミングの概念であったりと多岐にわたり課題が指摘されている．

具体的には，まず文法が挙げられる．プログラミングには様々な言語があり，特定の命令を組み合わせてコンピュータへの指示を記載する．この記載する言語の文法が誤るとエラーとなり実行が行えなくなるが，この文法について正しく記載ができないという課題が指摘されている [1][2]．文献 [1] では，この課題の影響について，学習者の能力を把握するための出題を行う場合も文法的な内容が含まれるため，文法的な不理解が原因で指導者側が学習者の要素理解について正確に把握することが難しくなる，と言う点を挙げており，学習において広い範囲に影響を与えるとしている．しかしながら，別の文献ではこの文法については学習当初は誤りが散見され課題となるが，それは解決までに時間が掛からない課題であり，また，時間とともに減少するため大きな問題にはならないとしている．むしろ，文法的なエラーより実行するまで誤りに気づけない意味的な矛盾を含む誤りの方が問題であると指摘する研究もある [3].

プログラミングの誤概念について調査した報告によると，もっとも基礎的な要素である反復処理と値の出力においても誤概念を持つ学習者がいるため，基礎的な要素の理解自体も容易ではないことが示されている [4]．また，要素の組み合わせに関しては，当然ながら一つの要素を

理解する事よりもより難しく優位にその差が示される報告がある [5]．さらに，要素自体よりもプログラミングコード全体の把握やその構成をする力の習得が難しいという指摘もある [6]．

その他にも，プログラミングが実行される環境であるコンピュータシステムに関連する知識としてメモリやポインタという概念が十分に理解できておらず，この理解が不十分なためプログラミングに支障が出るという指摘がある [6][7]．また，学習者は指導者から見て自身の能力を過信して，十分に理解していなものについても理解したと見なす報告もある [6]．

### 2.1.2 理解多様性の課題

もしも，要素の理解が容易ではないとしても，学習者によりつまづく点が共通であったり誤概念の種類が少ない場合は，それに応じた対応が可能となる．しかしながら，プログラミングに関する報告としてはそのようにはならず，むしろ理解が学習者により異なるという多様性のある理解になってしまう点が課題として存在する．

文献 [4] によると，単純な要素であっても複数の誤概念の型が示されている．また，文献 [8] によると，単なる要素の理解であっても様々な理解があり，例えば分岐処理の場合，「２つに１つを選ぶ」「避けて通る」「２つに別れる」など多様な解釈が存在していると指摘がある．その他の例としても，メモリの理解についても，同様に誤概念が多様であり，学習者ごと様々な方略によって回答がなされているという指摘がある [7]．その他にも，理解後の知識の形成という観点でも，学習者が理解するに従い能力が別れて行くという指摘がある [5]．

このような結果として，理解できる学習者は学習項目について先に進めるが，理解できない学習者は，そのままとなってしまうため，理解状態の差が学習者間で広くなり，出来る者と出来ない者の差が広がる結果となり単位取得失敗率が学習の 1,2 年目において 30% – 40% と非常に高いという指摘もある [9]．

### 2.1.3 動機づけの課題

理解できない内容があるとしても，継続して学習を続けることにより克服が可能な場合もあるが，プログラミングにおいては，この学習における動機づけも課題となる．様々なプログラミングの研究を纏めた文献においても脱落者（Dropout）の調査が一つの研究課題になる例として挙げられる通り，改善すべき課題となっている [10]．また，前項にも挙げたとおりであるが，文献 [9] によると，プログラミング教科の単位取得失敗率について高い状況である指摘があるように継続して学習を行うことが難しい側面がある．

### 2.1.4 知識応用の課題

知識を習得するだけではなくて，その応用も課題となっている．文献 [6] によると，要素の理解も問題ではあるが，その知識の適用が特に問題となっているという指摘があり，学習者や指導者へのアンケート結果としても，実用的な講義の効果が高いという回答が得られている．また，他の調査でも，プログラミングについて一定の理解をしているように見える学習者がいたとして，あるプログラミングコードを読んで出力結果を記載できたとしても，それより難易度の低いプログラミングコードを書き起こすことができないという指摘がある [11].

## 2.2 従来型学習方法の特徴と問題点

### 2.2.1 概要

プログラミング教育の改善のために様々な研究が発表されている．プログラミング研究に関する発表について５つの国際会議の発表内容を４年間にわたって集計分析をした調査によると，「能力や適正，理解状態の調査」（Ability/aptitude/understanding）が 40 % を占めており，次に「指導・学習方法やツール」（Teaching/learning/assessment techniques Teaching/learning/assessment tools）が 35 %となっている．その他には「教授理論やカリキュラム」（Teaching/learning theories & models Curriculum）が 6 %と続いている [12]．これを踏まえると，多くは「どのように理解しているか」が多くの調査対象になっており，次に「どのように指導/学習するべきか」と続いている．

「どのように指導/学習するべきか」という研究の方向性についてより細かく分類と整理をした調査である文献 [13] によると次のような分類となる．

**カリキュラム** 情報工学分野などの教育としてのプログラミング授業の位置づけなどについて各国の事例をもとに検討する研究などがある．

**教育方法** 指導者主導の方式や学習者主導の方式，また学習者のグループサイズなどの規模など教育方法の効果に関する調査．その他にも問題解決型学習などの教授方法も含む．

**言語** 代表的な言語としては C 言語，C++ 言語，JAVA 言語がある．これらはシステム開発にも広く使われているため教育にも使用される．他には，教育などの特殊用途に開発された言語である Python, Logo, Pascal なども使用される．

**ツール** アルゴリズムや処理の過程などを可視化した Visualize 環境や，プログラミングの自

動評価ツール，教育用の統合開発環境などが提案されている．

様々な学習方法が提案されており，各種方法の効果について文献を横断的に調査した研究によると，例えば media computation（画像や音声などのメディアの操作をするプログラムを通して学習する手法）を，ペアプログラミング（２人で交互にプログラミングし気づきや誤りを指摘する手法）で行う研究は，効果が高いと言う報告があるが，いずれも研究中であり調査者や調査条件などにより結果が変わる可能性が指摘されている [14]．文献 [14] によると，各種提案されている手法について一定の効果は認められるものの突出して効果的である手法はまだ報告されていない．また，別の報告によると，様々な学習方法があるがそのどれもが効果的であるものの，意図している方向が異なるために，それぞれ独自の方向のスキルが改善されている状態ですべてを満たすものではないと言う指摘がある [15].

次項では，提案されている学習方法について，その特徴と課題を記載する．しかしながら，提案手法の数が多いため，大きな分類に分割し，その中の代表的な手法について紹介する．大きな分類としては「理解型学習方法」と「経験型学習方法」に分けられる [16]．文献 [16] によると，これらは 1970 年代より変わらず対立構造として存在しており多くの「理解型学習方法」の支持者と一部の「経験型学習方法」の支持者がいるとされている．

### 2.2.2 理解型学習方法

もっとも一般的な学習方法として理解型学習方法があげられる．これは各要素について説明を行い，その要素の理解を深めるために学習した要素を含む応用問題を学習する．そして，その後は学習した要素の組み合わせの学習に進む．理解とその知識の演繹を基本とした流れの学習になる．文献 [17] は，従来型の教科書の中でも古くから存在しており，日本語版も 1994 年から出版されており，第 4 版まで改定がされる代表的なものの一つである．このような学習方法をとることにより，理解し自らコードを書けるようになる学習者も一定数存在するが，前節 2.1 にて記載したような課題があげられるため，新しいアプローチの研究がなされている．

近年多く提案されている手法の一つにプログラミングの可視化 (Visualize) による学習方法がある [12]．これは，プログラミングの一連の処理であるアルゴリズムを可視化して理解しやすくするアプローチであったり，日常生活では身近ではないプログラミングコード自体の表現を変更してブロックを繋げるような直感的なインタフェースにする方法であったり，プログラミングの処理状態について可視化を行う方法であったり，と理解向上のために可視化行う手法であり数多くのツールが提案されている [18]．この手法の学習効果としては，一定の効果を示

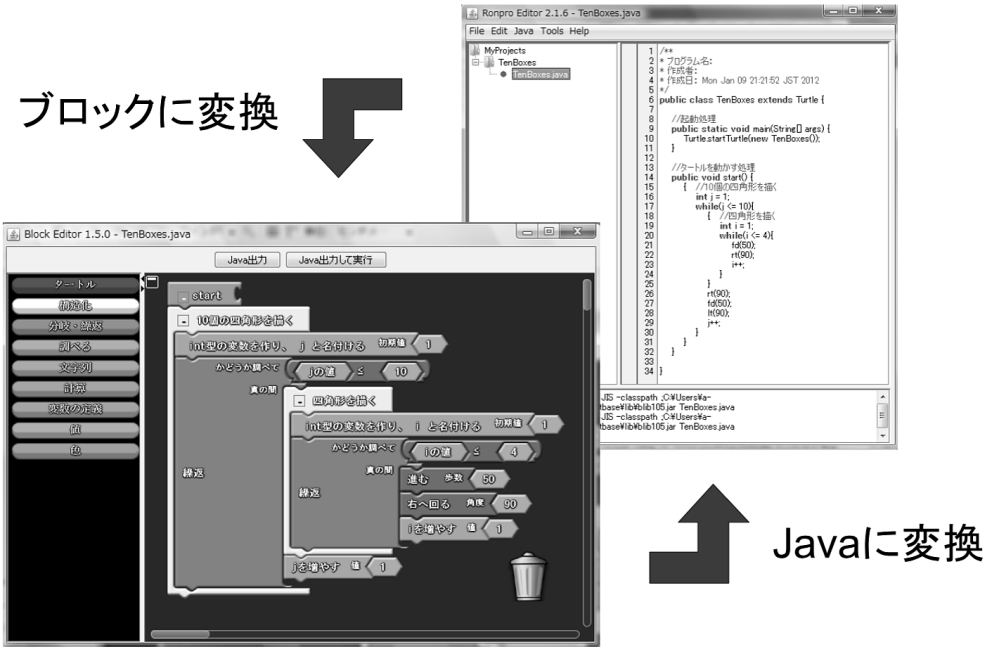


Fig. 2.1 Screen of a ”BlockEditor”[20].

す報告はあるものの，効果が見られないという報告もあるため有望な手法であるものの決定的ではない [12][19]．また，この手法（特にプログラミングコード自体の可視化）の課題としては，文字ベースのプログラミングへの移行に課題があるという指摘があり，解決策の模索も行われている [20]．この移行については研究により将来はスムーズな移行の可能性はあるが，移行自体に一定のコストがかかることに変わりはない．また，文字ベースと絵ベースの理解について，見せ方により差は大きくないとする報告もあり，可視化の表現の仕方にはまだ改善の余地はある状態である [21].

文献 [20] は，プログラミングコード自体の Visualize ツールの一つで BlockEditor であり，広く適用されているブロックベースのプログラミングツールと Java 言語へのコードの変換が可能であることが特徴である．画面を Fig. 2.1 に示す．同種のツールは様々作成されており [18]，文献 [20] 自体も日本で開発されたツールであるが，その他にもコード自体の Visualize ツールが海外のみならず，日本でも提案されているほど注目を集めている手法である [22].

その他の Visualize ツールとしては，実行中のコードの変数の状態について詳細に可視化する研究もある [23]．この研究は現在でも開発が行われていてシステムを無料で一般公開しており，言語も Python, Java, Ruby, C などと多数提供されている [24]．例として C 言語の実行中の画面を Fig. 2.2 に示す.

以上のような Visualize ツールの特徴としては，コードベースのインタフェースから，ブロッ

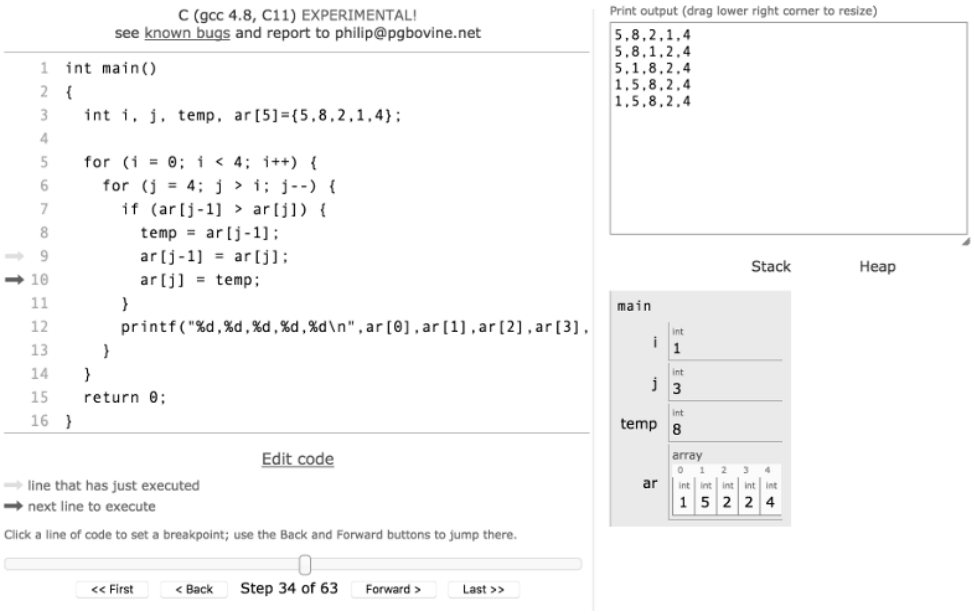


Fig. 2.2　Screen of a ”C Tutor”[24].

クベースになる手法であったり，通常では確認しづらい変数の状態を表示する手法であったりと，直感的に理解を向上させる手法である．そのため，要素会得と理解多様性の課題については従来型の教科書より改善が行われている．また，直感的にわかりやすい特徴のため，通常の開発環境と比較すると動機づけの課題の改善も行われている．しかしながら，本手法自体は複雑なコードや長いコードにおいては通常のプログラミング画面より情報が多いため，煩雑になり逆に扱いづらくなる点もあるため，応用問題への適用に課題があり，知識応用の課題の改善に関しては従来と比較して改善が見込めない．

### 2.2.3　経験型学習方法

学生の理解度についてその要因を調査した研究によると，投資した時間が大きな要因であるという報告がある [25]．また，別の研究では学習方法の効果を調査した結果として，ツールの要因より，学習時間の要因のほうが学習効果に与える影響として大きいという報告もある [26]．これらを踏まえると，プログラミングに多く触れる経験を通して得られる何らかの能力が重要であると考えらえる．

経験的な手法としては，多数の問題を解くドリル形式の学習が代表的ではあり，多数の問題を作成する出題側の負荷を軽減する自動問題作成に関する手法の研究が行われている [27][28]．ただし，単純なドリル形式の場合は，その学習効果自体について科学的な研究が行

われることが少なく，学習効果の確認や理論的な効果の裏付けがなく，どのような学習形式でどのような出題内容が最適であるかについて明確になっていない．

問題を解きながら学ぶという方針の手法を取る文献 [29] においては，認知的徒弟制理論（Cognitive Apprenticeship）をプログラミング教育に適用し，説明より課題を解くことを優先した学習方法を取り，かつ継続的なフィードバックを返すことにより，単位取得率を向上させて脱落者数を低下させる事が可能であるという報告がある．この手法は，教授の時間を削減して，宿題形式での問題解決学習を増加させる方式を取っている．類似した手法として，文献 [30] によると，授業で学習した内容を反復的に学習するために，宿題として読み学習や書き学習方式の課題を与える方法がある．なお，この手法においてはプログラミングでよく利用されるコードのまとまりを常套句として，集中的に学習を行う方針を採用している．いずれの手法も効果として理解度の向上，特に低い得点群の底上げ効果が期待される結果となり，要素会得や理解多様性，動機づけの課題について効果が見込める．また，応用的な内容の学習にも適用は可能であると考えられるため，知識応用の課題に対しても一定の効果が期待できる．しかしながら，この手法は宿題という形で，学習時間自体を拡張している方針であり，他の手法と比較して同一条件でどのような効果になるか調査は行われていない．前者のフィードバックを返す手法については，これについては教授側の負担が大きくなるという課題がある．また，どのように反復学習を行うかについては理論的な裏付けがなく，本手法を適用する場合に適切な適用をするための基準がない点も課題となる．

その他には，シリアスゲームやゲーミフィケーションと呼ばれる学習方法がある．これは，必ずしも経験を積むこと自体を意図したものではないが，テレビゲームのような見せ方をする学習教材を用いることで，学習効果を高める目的の教材である [31][32][33]．ゲームの内容により，反復的に学習を行う要素を含むため，単なる理解よりは経験を身につけるアプローチと見なすことができるが，学習効果としては報告により異なり明確に現れているわけではないが，モチベーションの向上につながるという報告があるため，動機づけの課題に対する効果は期待できる．

その他の経験を重視した学習方法としては，問題解決型学習（Problem based learning）やプロジェクト学習（Project based learning）がある．問題解決型学習は，医療分野の教育において始まった手法であり [34]，教科書的な知識のみでは現実の問題に対応できないという課題を解決するために，具体的な問題を設定し，それを解決する過程で実用的な力を身につける手法である [35]．プロジェクト学習も実際のプロジェクトを通して実用的な力を身につける手法であり，それぞれ様々な分野に応用がなされていて，プログラミング分野の学習にも応用がされ

ている [36][37][38]．文献 [6][11] などでは学習者の応用力が大きな問題であるとして，問題解決型の学習の必要性が指摘されている．文献 [13] の指摘では，問題解決型の学習は応用力向上が期待される手法であり，研究対象として注目を集めているという指摘がある．問題解決型学習は，要素会得や知識応用の課題について効果が見込める．また，応用的な内容の学習が中心であるため発展的な授業においては効果が期待できる．一方で本手法は，基礎学習が完了した後の応用力を向上する目的で適用されるため，基礎要素の学習には適用しづらいという課題が存在する．また，その他の課題としては，人員的なサポートを必要とすることから大人数講義の対応や指導補助人員などのリソースの準備が大変であるという指摘，難易度が高い学習となるためについて行けない学習者が発生するという指摘 [39] や，授業で適用するには時間がかかるため効果には限界があるという指摘もある [13].

## 2.3 まとめ

プログラミング学習の課題は様々な点が指摘されるが，大きな分類としてまとめると，「要素会得の課題」「理解多様性の課題」「動機づけの課題」「知識応用の課題」がある．

従来提案されてきた学習手法は，知識の理解とその演繹が主要なアプローチであった．しかしながら，従来のアプローチではまだ十分な結果が出ていない．一方で経験と言う要素も重要であるとする立場もあり，経験を重視すると見なせる学習方法もあるが，経験を重視する関係上，学習時間が多く必要になってしまう課題や指導側の負担が大きい事や，手法により導入部分でのみの適用となっているなど全ての課題に対応できる手法はない．

この関係の概要を Table. 2.1 に示す．

## 参考文献

[1] Andrew Luxton-Reilly and Andrew Petersen. The compound nature of novice programming assessments. In Proceedings of the Nineteenth Australasian Computing Education Conference, ACE '17, pages 26–35, New York, NY, USA, 2017. ACM.

[2] Arto Vihavainen, Juha Helminen, and Petri Ihantola. How novices tackle their first lines of code in an ide: Analysis of programming session traces. In Proceedings of the 14th Koli Calling International Conference on Computing Education Research, Koli Calling '14, pages 109–116, New York, NY, USA, 2014. ACM.

[3] Amjad Altadmri and Neil C.C. Brown. 37 million compilations: Investigating novice pro-

Table. 2.1 Comparison between conventional methods.

| Method | Target | | Under–standing | Diversity of under–standing | Motivation | Applied skill | Remarks |
|---|---|---|---|---|---|---|---|
| | Intro–duction | Appli–cation | | | | | |
| Conventional[17] | ✓ | ✓ | | | | | |
| Visualize tool[20][23] | ✓ | | ✓ | ✓*1 | ✓ | | |
| Extreme apprenticeship [29] | ✓ | ✓*1 | ✓ | ✓ | ✓ | | Learning time Demanding to instructor |
| Repetitive Lessons [30] | ✓ | ✓*1 | ✓ | ✓ | | ✓*1 | Learning time Theoretical background |
| Gamification[32] | ✓ | ✓*1 | ✓*1 | | ✓ | | |
| Problem based learning[36] | | ✓ | ✓ | | | ✓ | Large class Learning time Demanding to instructor |

*1 Limited improvement

gramming mistakes in large-scale student data. In Proceedings of the 46th ACM Technical Symposium on Computer Science Education, SIGCSE '15, pages 522–527, New York, NY, USA, 2015. ACM.

[4] Takayuki Sekiya and Kazunori Yamaguchi. Tracing quiz set to identify novices' programming misconceptions. In Proceedings of the 13th Koli Calling International Conference on Computing Education Research, Koli Calling '13, pages 87–95, New York, NY, USA, 2013. ACM.

[5] M. Yamamoto, T. Sekiya, and K. Yamaguchi. Relationship between programming concepts underlying programming skills. In 2011 International Conference on Information Technology Based Higher Education and Training, pages 1–7, Aug 2011.

[6] Essi Lahtinen, Kirsti Ala-Mutka, and Hannu-Matti Järvinen. A study of the difficulties of novice programmers. SIGCSE Bull., 37(3):14–18, June 2005.

[7] Lisa C. Kaczmarczyk, Elizabeth R. Petrick, J. Philip East, and Geoffrey L. Herman. Identifying student misconceptions of programming. In Proceedings of the 41st ACM Technical Symposium on Computer Science Education, SIGCSE '10, pages 107–111, New York, NY, USA, 2010. ACM.

[8] 長谷川 聡 , 山住 富也. プログラミング教育と学習者のイメージ形成 (その 2). 名古屋文理短期大学紀要, 23:9–14, apr 1998.

[9] Theresa Beaubouef and John Mason. Why the high attrition rate for computer science students: Some thoughts and observations. SIGCSE Bull., 37(2):103–106, June 2005.

[10] Syahanim Mohd Salleh, Zarina Shukur, and Hairulliza Mohamad Judi. Analysis of research in programming teaching tools: An initial review. Procedia - Social and Behavioral Sciences, 103:127 – 135, 2013. 13th International Educational Technology Conference.

[11] Raymond Lister, Elizabeth S. Adams, Sue Fitzgerald, William Fone, John Hamer, Morten Lindholm, Robert McCartney, Jan Erik Moström, Kate Sanders, Otto Seppälä, Beth Simon, and Lynda Thomas. A multi-national study of reading and tracing skills in novice programmers. SIGCSE Bull., 36(4):119–150, June 2004.

[12] Judy Sheard, S. Simon, Margaret Hamilton, and Jan Lönnberg. Analysis of research into the teaching and learning of programming. In Proceedings of the Fifth International Workshop on Computing Education Research Workshop, ICER '09, pages 93–104, New York, NY, USA, 2009. ACM.

[13] Arnold Pears, Stephen Seidman, Lauri Malmi, Linda Mannila, Elizabeth Adams, Jens Bennedsen, Marie Devlin, and James Paterson. A survey of literature on the teaching of introductory programming. SIGCSE Bull., 39(4):204–223, December 2007.

[14] Arto Vihavainen, Jonne Airaksinen, and Christopher Watson. A systematic review of approaches for teaching introductory programming and their influence on success. In Proceedings of the Tenth Annual Conference on International Computing Education Research, ICER '14, pages 19–26, New York, NY, USA, 2014. ACM.

[15] M. Lykke, M. Coto, S. Mora, N. Vandel, and C. Jantzen. Motivating programming students by problem based learning and lego robots. In 2014 IEEE Global Engineering Education Conference (EDUCON), pages 544–555, April 2014.

[16] L. Rolandsson. 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. In 2013 Learning and Teaching in Computing and Engineering, pages 220–223, March 2013.

[17] Herbert Schildt. Teach Yourself C. McGraw-Hill Osborne Media, 1997.

[18] Juha Sorva, Ville Karavirta, and Lauri Malmi. A review of generic program visualization systems for introductory programming education. Trans. Comput. Educ., 13(4):15:1–15:64, November 2013.

[19] Thomas W. Price and Tiffany Barnes. Comparing textual and block interfaces in a novice programming environment. In Proceedings of the Eleventh Annual International Conference

on International Computing Education Research, ICER '15, pages 91–99, New York, NY, USA, 2015. ACM.

[20] 松澤 芳昭 , 保井 元 , 杉浦 学 , 酒井 三四郎. ビジュアル-java 相互変換によるシームレスな言語移行を指向したプログラミング学習環境の提案と評価. 情報処理学会論文誌, 55(1):57–71, jan 2014.

[21] Stéphane Conversy. Unifying textual and visual: A theoretical account of the visual perception of programming languages. In Proceedings of the 2014 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software, Onward! 2014, pages 201–212, New York, NY, USA, 2014. ACM.

[22] 中西 渉 , 辰己 丈夫 , 西田 知博. Penflowchart によるプログラミング導入教育の評価. 情報処理学会 研究報告コンピュータと教育（CE）, 2013-CE-121(9):1–7, 2013.

[23] Philip J. Guo. Online python tutor: Embeddable web-based program visualization for cs education. In Proceeding of the 44th ACM Technical Symposium on Computer Science Education, SIGCSE '13, pages 579–584, New York, NY, USA, 2013. ACM.

[24] C tutor. http://www.pythontutor.com/c.html.

[25] Amir Kirsh and Iris Gaber. Satisfaction, time investment and success in students' programming exercise. In Proceedings of the Programming Experience 2016 (PX/16) Workshop, PX/16, pages 9–20, New York, NY, USA, 2016. ACM.

[26] G. Silva-Maceda, P. David Arjona-Villicaa, and F. Edgar Castillo-Barrera. More time or better tools? a large-scale retrospective comparison of pedagogical approaches to teach programming. IEEE Transactions on Education, 59(4):274–281, Nov 2016.

[27] Shuktika Jain. Automated generation of programming language quizzes. In Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, pages 1051–1053, New York, NY, USA, 2015. ACM.

[28] N. Funabiki, Y. Korenaga, Y. Matsushima, T. Nakanishi, and K. Watanabe. An online fill-in-the-blank problem function for learning reserved words in java programming education. In 2012 26th International Conference on Advanced Information Networking and Applications Workshops, pages 375–380, March 2012.

[29] Arto Vihavainen, Matti Paksula, and Matti Luukkainen. Extreme apprenticeship method in teaching programming for beginners. In Proceedings of the 42Nd ACM Technical Symposium on Computer Science Education, SIGCSE '11, pages 93–98, New York, NY, USA,

2011. ACM.

[30] 多田 知正 , 丸田 寛之. プログラミング教育における反復学習を採り入れた授業方式. 京都教育大学紀要, (116):123–134, mar 2010.

[31] T. Mitamura, Y. Suzuki, and T. Oohori. Serious games for learning programming languages. In 2012 IEEE International Conference on Systems, Man, and Cybernetics (SMC), pages 1812–1817, Oct 2012.

[32] D. C. Cliburn. The effectiveness of games as assignments in an introductory programming course. In Proceedings. Frontiers in Education. 36th Annual Conference, pages 6–10, Oct 2006.

[33] K. C. Yeh and W. F. Chen. Work in progress #x2014; using a computer gaming strategy to facilitate undergraduates' learning in a computer programming course: An experimental study. In 2011 Frontiers in Education Conference (FIE), pages S4H–1–S4H–2, Oct 2011.

[34] Howard S. Barrows. Problem-based learning in medicine and beyond: A brief overview. New Directions for Teaching and Learning, 1996(68):3–12, 1996.

[35] Erik de Graaff and Anette Kolmos. History of problem-based and project-based learning, pages 1–8. Sense Publishers, 2007.

[36] Esko Nuutila, Seppo Törmä, Päivi Kinnunen, and Lauri Malmi. Learning Programming with the PBL Method — Experiences on PBL Cases and Tutoring, pages 47–67. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[37] Jackie O'Kelly and J. Paul Gibson. Robocode & problem-based learning: A non-prescriptive approach to teaching programming. SIGCSE Bull., 38(3):217–221, June 2006.

[38] Theodora Koulouri, Stanislao Lauria, and Robert D. Macredie. Teaching introductory programming: A quantitative evaluation of different approaches. Trans. Comput. Educ., 14(4):26:1–26:28, December 2014.

[39] KINNUNEN Päivi and MALMI Lauri. Problems in problem-based learning - experiences, analysis and lessons learned on an introductory programming course. INFORMATICS IN EDUCATION, 4(2):193–214, 2005.

# 第 3 章

# 経験により得られる能力と知識形態，推論，並列的な能力の使用に関する従来研究の知見

## 3.1　経験が学習や理解に与える影響に関する４つの観点

経験により得られる能力などについて，認知科学分野の研究を中心として知見の紹介を行う．紹介する内容については，以下の４つの観点に基づいた内容を挙げる．

**経験により得られる能力が存在するのか?**　経験により得られる能力の知見を挙げる
**どのような構造となっているのか?**　柔軟な知識形態の知見を挙げる
**理解→演繹とは仕組みが異なるのか?**　演繹も含む代表的な推論機構を挙げる
**理解だけでは不十分か (両方必要か)?**　同時並列的に機能する複数の能力を挙げる

## 3.2　経験により得られる能力

### 3.2.1　概要

単なる理解とは異なる経験により得られる能力について，認知科学研究の知見を記載する．初めに，従来手法が理解を重視すると言う点に対する能力として，一般的な講義などは受講せずとも経験により一定の理論を構築することができる人間の能力として素朴概念を記載する．次に，経験を重ねることで特定の能力について向上をするが，特に経験を重ねた上で得られる特化した能力である熟達化と，その特化した能力の狭い転移範囲である領域固有性について記

載する．

### 3.2.2 素朴概念/誤概念

素朴概念とは，体系的・科学的に学んだ知識以外で，日常生活などにより身につけた知識であり，一定の範囲であれば矛盾せず一貫したものである．

より詳細な分野として Naive mathematics や Naive Phisics（素朴物理学）として次のように説明される．

> Whether or not schooling is offered, children and adults all over the world develop an intuitive, naive mathematics. As long as number-relevant examples are part of their culture, people will learn to reason about and solve addition and subtraction problems with positive natural numbers.

[1, 575 ページ]

> Naive physics refers to the commonsense beliefs that people hold about the way the world works, particularly with respect to classical mechanics.

[1, 577 ページ]

数学的能力を調査した研究によると，日常的に商品を販売する経験をしている人間（研究では牛乳の販売者）が，学校などで習う掛け算などの数式とは別の能力を用いて日々の計算を行っていることが示されている [2]．この例では，1 ケースに牛乳が 16 個収納される場合，35 個販売する際には 2 ケースと 3 個の牛乳を渡せば良いという計算を瞬時に行う．料金の支払いも同様のことが発生し，日々の作業の負担を軽減するための認知的な能力が発達していることになる．これは学校で習う体系的な能力とは別のものであり特定の分野に特化した能力であるとされる．

素朴概念は，日常の経験を積み重ねる中で組み上げられた理論であり，強固な知識となっている．しかしながら，直感的に把握している内容の積み重ねであるため，しばしば科学的には厳密には誤った知識となることから，誤概念と並んで記載されることもある．成人においても，特に物理学でこのような誤概念が顕著であり，簡単な慣性の問題でも誤ることが指摘されている [3]．また，このような素朴概念/誤概念を正そうとする研究があるものの，強力な知識となっているため，その変更は一時的には変化が見られても時間の経過とともに元に戻るとい

う指摘もあり，容易ではない [4].

プログラミングに関しても誤概念は研究対象となっており [5]，初歩的な内容である反復処理においても誤概念を形成してしまう点から見ても，一般的な講義を受けた大学生も何らかの素朴概念を保有しており，しばしば誤概念になると言う事がわかる．

### 3.2.3　熟達化

熟達化とは，大量の経験によって特定の分野において特別な能力を発揮する力を得る（熟達者になる）ことを指し，次のように説明される．

> Expertise refers to the mechanisms underlying the superior achievement of an expert, that is, “ one who has acquired special skill in or knowledge of a particular subject through professional training and practical experience ”

[1, 298 ページ]

熟達化によって得られる能力の具体例として，チェスや囲碁，将棋などのボードゲーム熟達者の盤面記憶スキルがあげられる [6][7]．例えば，将棋の研究では，初心者であれば開始から 60 手ほど進んだ盤面の記憶に 5 分弱程度かかるにもかかわらず，熟達者になると 20 秒程度で記憶することが可能になる．これは，チャンクなどと呼ばれる駒配置のパターンを記憶の塊を対象の経験から持っており，かつそれを効果的に認識，保持する力を得ていると考えられている．このような力は，例えば将棋のルールを正確に理解したりするだけで向上するわけではなく，対局などを通した大量の盤面の状況を経験した中から生まれてくるものである．なお，同様の力はプログラミングにおいても見られるという報告がある [8].

熟達者は，単純に記憶を沢山集めただけではなく，特定の分野の問題に対応できる形として形成された知識を有していると考えられている．熟達者が問題を解いてゆく過程を調査した研究によると何らかの問題に対処する場合，単純に記憶から何らかの知識を引き出してそれを適用するだけではなく，過去の経験から類似した状況を挙げて，そこから現在の状況においてどのようなことが発生しうるかなどの予測を立てることを行っているという研究結果がある．[1, 299 ページ]

また，このような熟達化については，熟達化した能力が自動的・非自覚的に使用されることが特徴である．文献 [9] は，幼児の言語習得過程を調査した研究であるが，言語の能力が発達するに従い，認知的な干渉を起こすタスクにも対応できるようになるなど，より少ない資源の

消費で作業が行えることがわかっている．

### 3.2.4　領域固有性

領域固有性とは，会得された認知能力について，様々な分野に応用できる普遍的な力ではなく，特定の分野においてのみ適用される，推論の仕方や知識構造のことを指し，次のように説明される．

> Cognitive abilities are domain-specific to the extent that the mode of reasoning, structure of knowledge, and mechanisms for acquiring knowledge differ in important ways across distinct content areas.

[1, 238 ページ]

この領域固有性は熟達化された能力の狭い適用範囲をさし示す文脈にも使用される．チェスの盤面を記憶する研究によると，ランダムに並べられた盤面の記憶においては，初心者と同じレベルの記憶力しか発揮できないという結果がある．また，単純な数字列の記憶も高い結果は示されない．チェスの熟達者が盤面を正確にそして高速に記憶できる能力を有しているが，これは単純に一般的な記憶力が高いものが熟達者になるわけでもなく，またそのような一般的な記憶力がチェスの経験で向上しているわけではないことを示している [6][1, 239 ページ].

プログラミングにおいても，過去の経験がプログラミング学習に与える影響の調査結果として，何らかの言語のプログラミング経験がある場合は，そうでないものに比べて優位に成績が良いことが示されており [10]，プログラミングという枠組みの中で共通した領域固有の力があることが示唆されている．しかしながら，数学の成績がプログラミング科目の成績と相関があるという指摘があるため，完全に独立した知識ではないと見られる [11].

## 3.3　柔軟な知識形態

### 3.3.1　スキーマ

柔軟な知識形態に関するスキーマ理論を記載する. スキーマとは経験を積み重ねる中で得られる柔軟な知識の構成や枠組みであり，次のように説明される．

> スキーマは変数をもつことにより様々な対象の表象に利用できる. また利用可能ではない情報については, 典型的な値であるデフォルト値を割り当てることにより, 表象を 豊

かにすることができる. またスキーマに表象された情報は各種の関係によって相互に関連づけられる. 従って, スキーマは単純な属性リスト表現に比べてはるかに豊かな表現力をもっている. スキーマは上位, 下位のスキーマと階層的な関係によって結びついており, これらの間には属性の継承関係があるとされる.

また，上記の説明に補足として，次のような説明を付加される.「わずかに異なった多種の経験を抽象化することで形成される」[12].

スキーマのような柔軟な知識表現が可能な特徴から，応用力を向上させる目的で小学生を対象として算数分野の教育に適用されている例がある [13][14][15]．これらは学習対象の知識を抽象化する目的で，具体的には比率や割合の計算の学習に適用されているが，結果については，良い影響があるという報告とそうではないという報告がなされており一貫していないものの，スキーマが教育に適用できる可能性を示す参考例と見なせる.

## 3.4　代表的な推論機構

### 3.4.1　概要

従来的な学習手法が理解とその知識の演繹を重視しているが，本研究はそれ以外の推論も関連すると考える. そこで認知科学研究の対象となる代表的な推論機構の 3 つである「演繹」「帰納」「類推」について記載する [16].

### 3.4.2　演繹

一般的な前提から個別の結論を得る演繹推論については，もっとも古くから研究が行われてきた領域であり，研究が行われてきた方向としては三段論法のような明確な論理であっても人は様々なバイアスを持っており誤った結論を導き出すという知見が得られている．具体的な例としては A は B であるという命題に対して，B は A であるという転換バイアスや，仮説を確証する推論を行い，仮説を反証する推論を行わない傾向がある確証バイアスなどがある.

### 3.4.3　帰納

個々の具体的な要素から一般的な前提を導き出す帰納について，認知科学的な研究は多くはないもののカテゴリーに関連した研究は一定数存在しており，知見としては次のようなものがある.

**「前提の典型性」効果**　ある要素から別の要素にその特徴を一般化する場合（特殊論証と呼ばれる）には，前提の要素に特徴が類似している，すなわちより近いカテゴリーに属すものと思われるほど，より妥当な一般化と判断する傾向がある．例としては，「まぐろの血液中の鉄分の濃度は人間よりも高い」という前提があるとして，それを別の要素に一般化する場合は，「かつおの血液中〜」という結論の方が「ひらめの血液中〜」という結論よりもより妥当に感じる傾向がある．

**「前提の多様性」効果**　個々の要素から一般化を行う際（一般論証と呼ばれる）には，個々の要素の特徴に多様性があるほど，すなわちより異なる特徴があるものが集まるほど，より妥当な一般化と判断する傾向がある．例としては，２つの要素から一般的な事項を導き出す帰納を行う際に，「まぐろの血液中の鉄分の濃度は人間よりも高い」という要素があるとして，それに並ぶ要素が「ひらめの血液中〜」である方が「かつおの血液中〜」という要素が並ぶ方がより妥当に感じる傾向がある．

### 3.4.4　類推

類推とは，過去に経験した知識を元にして（ベースと呼ばれる），それを現在の問題（ターゲットと呼ばれる）を理解や説明し解決する推論の形式である．これは類似という特徴から未知の問題に対応できるため，人間の柔軟な思考解明の鍵として 1980 年代から多く研究されている．典型的な例としては，電気回路の電圧・電流・抵抗を水の落差・水流・水の抵抗物と見なす例である．

この類似という判断の基準としては機械的に考えると観点は様々あり収束させることが難しい．そこでこの類似については，様々な理論が提案されており，代表的なものとしては，構造に着目した類似判断を行うという考えや，構造だけではなく目的や状況に応じた類似検索の制約が働くという考えが示されている．また，近年では類似の制約である構造について，これが類推の過程で生成されるという考え方も示されており，人間の振る舞いについて多くが説明できるモデルが研究として展開されている．

## 3.5　同時並列的に機能する複数の能力

### 3.5.1　概要

経験により得られる能力が従来的な学習により得られる理解と演繹の能力とは別に存在するとして，それらが片方では不十分であり，両方とも必要であるか検討を行うため，同時並列的に機能する複数の能力を挙げる．人間の知能やそれを模倣した研究について，複数の要素が関連するという立場の研究を記載する．

### 3.5.2　2 つの思考様式

J. S. Bruner[*1]が提唱した考えに，2 つの思考様式がある [18]．これは，論理－科学的思考様式という論理的な思考と，ナラティブ様式という非論理的な思考の存在を指摘したものとなる．この様式の説明は後述するがその前に，その考えに関連した別の研究者の提案を先に記載する．Bruner は，Jean Piaget や L. S. Vygotsky といった心理学者の提唱を踏まえた上でそれを折衷，統合という手法をとったという見方 [18] や，Bruner 研究においては，Piaget や Vygotsky との比較研究が行われる [19] という点から見ても関連性が高いため，それぞれの主張について先に説明を行う．

なお，これらの研究者については古く，Vygotsky は 1930 年代の研究者で Piaget も同年代であり，Bruner も有名な著作である「教育の過程」は 1959 年であるように決して新しい研究者ではない．しかしながら，それぞれの研究者が現代においても研究の対象になっていることからわかるように [20] [21] [22][23]，心理学・認知科学分野に与えた影響は現在でも大きい．

Piaget は知能の発達段階説を主張した．これは，「感覚運動期」「前操作期」「具体的操作期」「形式的操作期」という段階に別れた発達が普遍的に行われるという考え方である．「前操作期」には理解できない物体の同一性や保存の概念（液体を形状 A の容器から形状 B の容器に移しても液体の量は変わらない事を理解できる事）が得られる「具体的操作期」は 6 歳から 10 歳頃に形成され，物体を使用した思考ではなく，文章を使用した抽象的な思考ができる「形式的操作期」は 11 歳から 15 歳頃に形成されるという主張である．これは，すべての領域にまたがる普遍的な知能の発展であるという解釈がなされ，現在支持を集めている領域固有な知

[*1] アメリカの教育心理学・認知科学研究者．1959 年に開催されたウッズホール会議 – アメリカの自然科学教育の改善の為に，全米科学アカデミーの呼びかけで 34 人の科学者，学者，教育者によって開かれた会議 – の議長を務めた．その成果をのちに出版した [17].

能という知見に合致しないと考えられる事から批判を受ける対象になっているものの，現代においてもこの理論を支持する研究者もいる [20]．この主張については，本研究の観点から見た注目すべき点は，物体を用いて思考する知能は，その後発達する論理的な思考が可能な知能の下位に属しており，新しい知能が古い知能に置き換わるという視点が含まれているという点である．

Vygotsky は，知能の発達において「生活的概念」と「科学的概念」という考えを主張した [24].「生活的概念」は，生活を通じて身につけた概念であり，正しく利用はできるが自覚がないため随意的に使用はできない概念である．具体的には子供が身につける「兄弟」という概念を挙げている．（子供は概念自体は正確に理解しているものの，言葉で正確に説明ができない場合がある）一方で「科学的概念」とは講義などを通じて体系的に学んだ概念を指し，随意的・自覚的に使用ができる．具体的には「アルキメデスの法則」など，言語的に定義され理解した概念である．この「生活的概念」と「科学的概念」が互いに作用しあい発達してゆくものであると，Vygotsky は主張した．この主張では「生活的概念」が一定の水準に到達すると，「科学的概念」の発達が可能になり，科学的概念の発達による概念の一般化により，「生活的概念」の構造に変化をもたらし，抽象的な思考が可能となる．と二つの関係性を主張している．本研究の視点で捉えると，二つの概念が置き換わるのではなく，互いに影響を及ぼし発達し変化するという点が注目すべき点となる．

Bruner の「２つの思考様式」の一つである「ナラティブ様式」とは，個人的な経験に紐づくような思考で，物語を扱うように特殊化された思考様式である．社会科学や人文学の説明に使用される思考様式という説明もある．「論理－科学的思考様式」とは，物理的な物事を扱うための客観的で体系化された思考様式である．数学や科学のように分析や論理的証明を行う思考である．Bruner はこれらの思考様式が，片方に支配されることはなく，また片方に還元されることもない不可分な関係であり相補的な関係と主張した [18][23]．本研究の観点からこれを捉えると，重要な点は，二つの思考がそれぞれ合流せずに別々に並列的に存在しており，相補的に機能するという点である．

### 3.5.3　冗長な認知システム

人間の認知活動を一般的にモデル化した場合は，始めに人が感覚器官による知覚し，そしてそれを受けて脳で思考や計画の立案する．最後に脳からの指令を受けて手足などによる動作という図式が一般的であり，要するには単一の回路が想定される．しかしながら，認知科学研究

が進むにつれて，人の行う処理は単一の回路や手順ではなく，並列的な回路による実行が可能なことがあることがわかっており，重層的，冗長な仕組みを持ち合わせているという見解がある [25]．人の柔軟な思考や行動が，そのような重層性や冗長性と関連しているという指摘があり [16] トップダウンな概念構造を用いた処理と並行して多数の局所的なボトムアッププロセスが処理されているというマルチエージェント的なモデルが提案されている．

実際に，人の動作を細かく見た研究によると，マイクロスリップと呼ばれる現象が確認されており，並列的な処理の根拠としてあげることができる [26]．スリップとは，脳に障害を負った患者などが，健常者であれば容易にこなせる身体的な動作に失敗することを指す用語であるが，実はこの研究の指摘によると，小さいながらも健常者であれ日常の行動の中に多数の小さなスリップが非自覚的に含まれていることがわかった．コーヒーを入れるタスクの観察実験を通して，手の動作のよどみや軌道の変更などが確認され，手を動かす前に完全にプランニングされてから指示が出るわけではなく，マルチエージェント的な仕組みが関連しているということが示された．

その他にも，人工知能分野において，人間のように柔軟に活動が可能なロボットの研究でも，並列的な処理の提案がなされている [27][28]．これは，サブサンプションアーキテクチャと呼ばれ，発表当時は批判を浴び，議論を巻き起こしたものの，その後家電製品への技術応用なども含めこのような考え方が浸透してきており，注目を浴びている [29]．これは，従来型の知覚–思考–指示が，ひとまとまりになった直列的な処理体系とは異なり，シンプルな処理を行う小さなエージェントが並列的に処理を行うものである．一つ一つは単純な処理しかできないものの，それらが並列的に動作することで結果として複雑な動きとして現れる創発性が特徴である．

### 3.5.4　ダイナミズム

近年の人間の思考の一つの新しい潮流であるダイナミズムがある [25]．認知科学的な思考の研究においては，その歴史として初めに一般的な知能やルールがあるというモデルの元に調査研究がなされてきた．しかし，そのような知能は見つからず，過去の経験に基づく領域固有な能力があることが知見として得られた．それを受けて，知識量を重要とみなす流れとなり，その構造が研究されてきた．しかしながら，知識についても無制限に保持できる保証はなく，また経験したもののみの対応しかできないわけではなく，人間は柔軟に振る舞えることを踏まえるとさらに新たなアプローチの研究が必要になってきた．

そこで，一つの方向性としてダイナミズムという考え方が示されており，これは思考を頭の中だけで行われるものではなく，内的資源と外的な環境（図，文章，他者，文化，社会など）とがダイナミックに相互作用を起こすという考え方となる．「類推」の項で触れたように構造などの一部の資源がダイナミックに生成されるという考え方が示されたり，生物学的なアプローチとしての認知神経学の知見としては，脳の著しい可塑性や，脳のさまざま部位間で見られる双方向結合による複雑な相互作用システムの研究などがある．

# 参考文献

[1] Robert Andrew Wilson and Frank C. Keil. The MIT Encyclopedia of the Cognitive Sciences. MIT Press, 2001.

[2] Magdalene Lampert. Knowing, doing, and teaching multiplication. Cognition and Instruction, 3(4):305–342, 1986.

[3] 吉野 巌 , 小山 道人. 「素朴概念への気づき」が素朴概念の修正に及ぼす影響-物理分野の直落信念と mif 素朴概念に関して-. 北海道教育大学紀要. 教育科学編, 57(2):165–175, feb 2007.

[4] 村田 佳苗 , 大仲 政憲. 小学校理科における素朴概念を活用した学習活動に関する基礎的研究–「水溶液の性質」を通して. 大阪教育大学紀要 第 5 部門 教科教育, 59(2):1–14, feb 2011.

[5] Takayuki Sekiya and Kazunori Yamaguchi. Tracing quiz set to identify novices' programming misconceptions. In Proceedings of the 13th Koli Calling International Conference on Computing Education Research, Koli Calling '13, pages 87–95, New York, NY, USA, 2013. ACM.

[6] William G. Chase and Herbert A. Simon. Perception in chess. Cognitive Psychology, 4(1):55 – 81, 1973.

[7] 伊藤 毅志 , 松原 仁 , ライエルグリンベルゲン. 将棋の認知科学的研究（1）－記憶実験からの考察. 情報処理学会論文誌, 43(10):2998–3011, oct 2002.

[8] Katherine B. McKeithen, Judith S. Reitman, Henry H. Rueter, and Stephen C. Hirtle. Knowledge organization and skill differences in computer programmers. Cognitive Psychology, 13(3):307 – 325, 1981.

[9] 高橋 登. 入門期の読み能力の熟達化過程. 教育心理学研究, 41(3):264–274, 1993.

[10] Dianne Hagan and Selby Markham. Does it help to have some programming experience before beginning a computing degree program? SIGCSE Bull., 32(3):25–28, July 2000.

[11] Divna Krpan, Saa Mladenovi, and Marko Rosi. Undergraduate programming courses, students ’ perception and success. Procedia - Social and Behavioral Sciences, 174:3868 – 3872, 2015. International Conference on New Horizons in Education, INTE 2014, 25-27 June 2014, Paris, France.

[12] 白水 始 , 三宅 なほみ. 認知科学的視点に基づく認知科学教育カリキュラム : 「スキーマ」の学習を例に. 認知科学 = Cognitive studies : bulletin of the Japanese Cognitive Science Society, 16(3):348–376, sep 2009.

[13] Milan Hejn, Jana Slezkov, and Darina Jirotkov. Understanding equations in schema-oriented education. Procedia - Social and Behavioral Sciences, 93(Supplement C):995 – 999, 2013. 3rd World Conference on Learning, Teaching and Educational Leadership.

[14] Asha K. Jitendra, Jon R. Star, Kristin Starosta, Jayne M. Leh, Sheetal Sood, Grace Caskie, Cheyenne L. Hughes, and Toshi R. Mack. Improving seventh grade students ’ learning of ratio and proportion: The role of schema-based instruction. Contemporary Educational Psychology, 34(3):250 – 264, 2009.

[15] Asha K. Jitendra and Jon R. Star. An exploratory study contrasting high- and low-achieving students’ percent word problem solving. Learning and Individual Differences, 22(1):151 – 158, 2012.

[16] 鈴木 宏昭. 人間の推論 (「認知科学」〔第 6 回〕). 人工知能学会誌 = Journal of Japanese Society for Artificial Intelligence, 16(6):858–865, nov 2001.

[17] JEROME S. BRUNER. The Process of Education. HARVARD UNIVERSITY PRESS, 1960.

[18] 今井 康晴. ブルーナーのナラティブ論に関する一考察. 広島大学大学院教育学研究科紀要. 第一部, 学習開発関連領域, (59):51–57, dec 2010.

[19] 今井 康晴. ブルーナーにおける幼児教育・保育論の展開に関する一考察 : 教育に関わる著作を中心に. 学習開発学研究, (7):29–35, 2014.

[20] 中垣 啓. ピアジェ発達段階論の意義と射程. 発達心理学研究, 22(4):369–380, 2011.

[21] 田島 充士. 言葉の理解およびその教育可能性をヴィゴツキー・内言論から捉える : スタニスラフスキー・ポドテキスト論を補助線として. ヴィゴツキー学 = Vygotsky studies, (4):45–57, nov 2016.

[22] Jonathan M. Adler. Two modes of thought: The narrative/paradigmatic disconnect in the bailey book controversy. Archives of Sexual Behavior, 37(3):422–425, Jun 2008.

[23] 嶋口 裕基. ブルーナーの教育論における「構造」の再検討-「論理-科学的様式」の教育方法としての「構造」のために-. 早稲田大学大学院教育学研究科紀要, 19(1):191–201, 2011.

[24] L. S. Vygotski and 翻訳：柴田 義松. 思考と言語. 新読書社, 2001.

[25] 鈴木 宏昭. 思考のダイナミックな性質の解明へ向けて. 認知科学, 8(3):212–224, 2001.

[26] 鈴木 健太郎 , 三嶋 博之 , 佐々木 正人. アフォーダンスと行為の多様性 : マイクロスリップをめぐって (＜特集＞新システム論の視座-関係性と多様性の回復を求めて-(その 2)). 日本ファジィ学会誌, 9(6):826–837, 1997.

[27] R. Brooks. A robust layered control system for a mobile robot. IEEE Journal on Robotics and Automation, 2(1):14–23, Mar 1986.

[28] 吉田 典晃. サブサンプションアーキテクチャを用いたロボットの制御. 日本ロボット学会誌, 11(8):14–19, nov 1993.

[29] 五味 隆志. サブサンプション・アーキテクチャ. 日本ファジィ学会誌, 7(5):909–930, 1995.

# 第 4 章

# 経験により得られる能力に着目したプログラミング学習方法の仮説

## 4.1　理論

### 4.1.1　経験により得られる能力

体系的な指導を受けずとも，経験を積むことで，素朴理論で説明されるように一貫した理論を自己の中に構築できる点を踏まえるとプログラミングにおいても説明を省略した状態から経験を積むことでも一定の理論を構築することが可能であると考えられる．素朴理論に関しては，説明を受けて理解をしたもののみ特別に得られる能力と言うわけではなく，素朴物理学などの重力や慣性などのように，同様の結論を多くの人間が得られる点から見ても要素会得と理解多様性の課題は発生せず，これの改善に利用できると考えられる．これをを踏まえ，概念を体系的な説明資料を用いて意味を理解する方針ではなく，多数の問題を回答しその場で正誤判定知る手法を取り，一定の時間内で経験を多くできる構成の教材とする．これにより，その出題範囲の知識にはなるが一定の体系だった知識が，従来方式よりは広い範囲の学習者に形成されると考えられる．

得られた能力は，熟達化理論として指摘されるように領域固有ではあるが，単なる知識の寄せ集め以上の問題に対処する構造として学習者に保持され，ルールや定義の理解だけでは解決できないような問題の解決が行われる能力が備わると考えられる．経験するに従いその経験の中の問題に対応できるように能力が特化し，それが単なる理解以上の問題解決が可能である点を踏まえると，要素会得（要素の深い理解という意味で）と理解多様性の改善に利用できると考えられる．そのため，広く多数の種類の問題形式を経験するのではなく，まずは特定の問題

形式について経験を積むことで学習対象の分野についての熟達化を促し，学習した問題形式の問題解決に即した形の概念を形成する手法が効果的であると考えられる．また，この手法のメリットとして，システム化が容易であるため，問題解決型学習のような多人数の学習における指導側の負担が発生しない．

また，領域固有な能力になる点を踏まえると，ブロックベースの学習方法は将来的な文字ベース環境への移行がいずれ必要になり，そのコストが一定は必要になることから好ましくはない．学習教材の表現方法であるが，Visualize 教材が現在は注目を集めるものの，領域固有な知識が得られてしまい，その転移にコストがかかることを踏まえると，学習の初期段階から，文字ベースの教材で通常開発と同じ表現をすることが好ましいと考える．

### 4.1.2 柔軟な知識形態

また，経験により構築された知識の形態としては，デフォルト値・変数化された知識（スロット）・階層構造 を特徴として持つスキーマ構造となると考えられる．このような状態になった場合は，それが柔軟な知識となることが想定される．またスキーマ理論自体が応用力の向上を期待して算数学習に適用を試みる研究がある点から見ても [1][2][3]，知識応用の課題に改善が期待できる．その為，学習問題の出題内容は，このデフォルト値とスロットを考慮して作成することが必要である．プログラミングとしてはどの部分が変数化の対象となるか調査は必要であるが，スキーマ的な知識の形成として，デフォルト値となりうる中心的な問題を多く出題し，かつ，中心的な問題からわずかに異なる問題を多数出題し，変数化を促す形式が好ましいと考えられる．

### 4.1.3 推論機構

演繹的な力も重要ではあるが，それだけでは不十分であると考えられる．プログラミング教育においては，一般的な定義を要素ごとに説明し，その後に，その要素の組み合わせを説明する流れとなる．要素の例としては，条件分岐処理や反復処理，データ形式などがあり，その組み合わせとしてソートアルゴリズムなどのアルゴリズムがある．これは要素や定義を理解することで，演繹的にその組み合わせも理解できるという前提に立つ教授方法であると言える．しかしながら，認知科学的な知見からわかる通り，人間はそこまで正確にルール通りに振る舞えないバイアスを持っている点からもこの方針だけで十分であると見なすことが難しい．

プログラミングにおいては，帰納は意図して教育に適用される例が少ないものの，反復学習

や問題解決学習などについては，経験を通して学ぶという考え方があるため，経験を通した一般化（様々なトライ&エラーの積み重ねから得られる知識の一般化）は起こると考えられる．

プログラミングにおいては，経験量が重要であるという指摘があることを踏まえると，過去の経験したプログラミングコードの知識が重要であるとみなせる．その場合，類推的な推論が行われており，過去の記憶にあるコードがベースになっていると考えることができる．具体的には，次のような単純なプログラミング知能のモデル化（単純モデル）の例と類推的な要素を含むモデル化の例（類推モデル）を比較する事で説明が可能である．初心者はプログラミングコードが組めない，そして，コードから出力結果を予測できない．そして熟達者は両方できる状態となる．プログラミングのコードはコンパイルと呼ばれる処理と，成果物の実行により，出力結果を得られる（説明を簡単にするため，リンクやインタプリタの概念は除外して説明している）．これを踏まえると，単純モデルの場合は，初心者にはコンパイルを行うコンパイラと成果物を実行する環境が頭に無い状態であり，熟達者は正確にコンパイラと実行環境を学習による知識のおかげで手に入れる状態となる，と言う説明ができる（単純モデル）．しかし，これには矛盾がある．なぜかというと，熟達者もケアレスミスをするからである．熟達者においても，単純な文法的なミスや，紛らわしい点の見落としが発生する．これを踏まえると，コードを把握するときに，一行一行，そして一文字一文字をすべて正確に把握できるわけではなく，過去の経験（ベース）を用いて，全体的な構造を類推によって把握しているという事が考えられる (類推モデル)．また，単純モデルのように演繹的に組み合わせを検討すると，組み合わせ数が発散してしまい深い思考ができなくなると考えられる．一方で類推モデルであれば，過去の知識とスキーマ的な構成となったスロットを持つ知識の適用のみで問題解決が可能であり，熟達化がなされている場合は自動的・非自覚的に使用が可能であるため，思考時のリソースを効率的に利用（ワーキングメモリを節約）でき，深い思考が可能となる．

以上の点から，演繹以外の推論機構もプログラミングにおいて経験により得られる能力の重要であるとみなせるため，類推的なプログラミング能力の調査は重要であり，類推のベースとなるデフォルト値の知識が重要なため特に反復的な学習が必要であると考えられる．

### 4.1.4 同時並列的に機能する複数の能力

従来教材では，理解とその演繹が中心であった．また，理解とは異なるアプローチとして，経験を重視した学習方法が提案されているが，それぞれ統合されるアプローチではなかった．

認知科学や発達心理学研究のこれまでの知見を簡単にまとめると，人間の柔軟な認知能力の

仕組みとして，単一の回路ではなく複数の回路で問題解決を図れるように冗長的，重層的な構成となっている．様々な知能が並列的に処理を行い，補完的に働く．その結果として，個のたし合わせ以上の複雑な処理として現れる創発性が見られるという特徴が見られる．このような点を踏まえると，「理解」「経験」のいずれも必要であり，一方だけでは柔軟な振る舞いが行えないことを踏まえると応用力に欠けると考えられる．反対に，「理解」「経験」双方を組み合わせることで，知識応用の課題の改善が期待できる．このように，経験型学習教材も理解型教材同様に重要である事が考えられる一方で実情はそのようになっていないことを踏まえると，確立されていない経験型教材の理論と方法の確立は重要である．

### 4.1.5 動機づけの対応

動機づけの対応として，ゲーミフィケーション手法に関連する研究や認知科学的な研究での，モチベーションの維持に関する知見を利用する．

文献 [4] によると，克服が期待される失敗は内発的動機付けの水準を高める効果があり，克服不能な確実な失敗は内発的動機付けを損なうとされる．これを踏まえると，難易度が高すぎる問題は好ましくなく，低難度の問題の出題を中心に考える．また，不正解時に次回の対応が可能に感じられるように誤回答の際に正答を表示することも有効であると考えられる．

その他に文献 [5] によると，やりたくない行為の軽減 (操作性の向上) がモチベーションの向上に関係するという指摘を踏まえると，学習時の作業について，単純な操作で学習可能な形態が好ましいと考えられる．

## 4.2 学習方法の仮説についてのまとめ

プログラミング学習において解決すべき課題としては，要素会得・理解多様性・動機づけ・知識応用がある．この改善をする学習方法について，前節の内容を踏まえた上で入門レベルのプログラミング学習向上に利用できる具体的な学習方法の仮説を Table. 4.1 にまとめる．

従来型の研究と本研究で提案した学習方法の仮説についてその特徴を Table. 4.2 に示す．この点から，従来の課題のそれぞれについて改善をした学習方法になる．ただし，今回の研究では入門レベルの改善を対象とするため，提案手法が発展授業に対しての適用が可能かどうかについては研究対象から除外する．

Table. 4.1 Required learning elements for the proposed hypothesis.

| Element | Understanding | Diversity of understanding | Motivation | Applied skill | Remarks |
|---|---|---|---|---|---|
| Less explanation<br>Immediate correct decision | ✓ | ✓ | | | |
| Continuous question in<br>a particular format | ✓ | ✓ | | | Large class |
| Character based<br>learning material | | | | | Domain<br>specificities |
| Schema based<br>questioning<br>Iterative questioning<br>Similar questions | | | | ✓ | |
| Combined use of<br>both understanding<br>and experience<br>based methods | | | | ✓ | |
| Low difficulties<br>Show correct answer<br>when incorrect<br>Simple operation | | | ✓ | | |

Table. 4.2 Comparison between conventional methods and the proposed hypothesis.

| Method | Target | | Under–<br>standing | Diversity<br>of under–<br>standing | Motivation | Applied<br>skill | Remarks |
|---|---|---|---|---|---|---|---|
| | Intro–<br>duction | Appli–<br>cation | | | | | |
| Conventional[6] | ✓ | ✓ | | | | | |
| Visualize tool[7][8] | ✓ | | ✓ | ✓*1 | ✓ | | |
| Extreme apprenticeship [9] | ✓ | ✓*1 | ✓ | ✓ | ✓ | | Learning time<br>Demanding to<br>instructor |
| Repetitive Lessons [10] | ✓ | ✓*1 | ✓ | ✓ | | ✓*1 | Learning time<br>Theoretical<br>background |
| Gamification[11] | ✓ | ✓*1 | ✓*1 | | ✓ | | |
| Problem based learning[12] | | ✓ | ✓ | | | ✓ | Large class<br>Learning time<br>Demanding to<br>instructor |
| Proposed method | ✓ | | ✓ | ✓ | ✓ | ✓ | |

*1 Limited improvement

## 4.3 仮説検証の方針

前節にて述べた方針の教材について効果を測定することを最終的には行う必要があるが，理論面の仮定も含めて検討すべき要素が多い．そのため，理論的な調査と実践的な調査をそれぞれ行った後に，結果を踏まえた学習方法を提案する方針とする．なお，本研究で行う実験は，東海大学「人を対象とする研究」に関する倫理委員会の承認を得た手続きに従う．

理論的な調査について，提案する仮説の根本に関わる，経験とプログラミング力，スキーマ，推論機構，の 3 点について仮説を設定して調査を行う．

(i) プログラミング力と経験は関連する
(ii) プログラミングの知識はスキーマ構造である
(iii) 演繹以外の推論が使用される

実践的な調査については，前節にて提案した仮説の学習要素について，実際の教材を用いた実験を実施して調査を行う．

1. 学習要素の説明は問題に取り掛かれる範囲まで（それ以上は行わない）
2. 学習問題の正誤即時判定
3. 特定形式の連続出題 (書き学習，または読み学習)
4. 文字ベースの教材
5. 出題内容はスキーマを意識した形態 (反復的な出題，類似問題)
6. 理解型教材と経験型教材の併用
7. 低難度の問題の出題
8. 単純な操作で学習可能
9. 誤回答の際に正答を表示

各要素の検討方針は次のとおりである．要素 1,2,4,8 は経験教材として主要な要素のため，実験に使用する学習方法に常に含める．要素 3 は，書き学習や読み学習など具体的な形式が未定のため，比較確認が必要となる．要素 5 は，理論的調査にて検証を行うため検討から除外する．要素 6 は，提案型教材と理解型教材の比較実験形式を取ることで，各学習方法の特徴を明確にして必要性を検討する．要素 7,9 は書きと読みの学習方法により配慮する点が異なるため，それぞれの条件において可能な範囲で適用する．

以上を踏まえ，仮説としては以下になる．

(a) 書きを中心とした経験型学習は理解型学習より学習効果が高い
(b) 読みを中心とした経験型学習は理解型学習より学習効果が高い

# 参考文献

[1] Milan Hejn, Jana Slezkov, and Darina Jirotkov. Understanding equations in schema-oriented education. Procedia - Social and Behavioral Sciences, 93(Supplement C):995 – 999, 2013. 3rd World Conference on Learning, Teaching and Educational Leadership.

[2] Asha K. Jitendra, Jon R. Star, Kristin Starosta, Jayne M. Leh, Sheetal Sood, Grace Caskie, Cheyenne L. Hughes, and Toshi R. Mack. Improving seventh grade students ' learning of ratio and proportion: The role of schema-based instruction. Contemporary Educational Psychology, 34(3):250 – 264, 2009.

[3] Asha K. Jitendra and Jon R. Star. An exploratory study contrasting high- and low-achieving students' percent word problem solving. Learning and Individual Differences, 22(1):151 – 158, 2012.

[4] 碓井 真史. 内発的動機づけに及ぼす外的報酬と認知的情報の効果. 社会心理学研究, 2(1):25–31, 1986.

[5] 中谷 智司 , 矢野 米雄. ロールプレイングゲームにおけるやる気の持続. 情報処理学会研究報告人文科学とコンピュータ（CH）, 1993(18):33–38, mar 1993.

[6] Herbert Schildt. Teach Yourself C. McGraw-Hill Osborne Media, 1997.

[7] 松澤 芳昭 , 保井 元 , 杉浦 学 , 酒井 三四郎. ビジュアル-java 相互変換によるシームレスな言語移行を指向したプログラミング学習環境の提案と評価. 情報処理学会論文誌, 55(1):57–71, jan 2014.

[8] Philip J. Guo. Online python tutor: Embeddable web-based program visualization for cs education. In Proceeding of the 44th ACM Technical Symposium on Computer Science Education, SIGCSE '13, pages 579–584, New York, NY, USA, 2013. ACM.

[9] Arto Vihavainen, Matti Paksula, and Matti Luukkainen. Extreme apprenticeship method in teaching programming for beginners. In Proceedings of the 42Nd ACM Technical Symposium on Computer Science Education, SIGCSE '11, pages 93–98, New York, NY, USA, 2011. ACM.

[10] 多田 知正 , 丸田 寛之. プログラミング教育における反復学習を採り入れた授業方式. 京都教育大学紀要, (116):123–134, mar 2010.

[11] D. C. Cliburn. The effectiveness of games as assignments in an introductory programming course. In Proceedings. Frontiers in Education. 36th Annual Conference, pages 6–10, Oct 2006.

[12] Esko Nuutila, Seppo Törmä, Päivi Kinnunen, and Lauri Malmi. Learning Programming with the PBL Method — Experiences on PBL Cases and Tutoring, pages 47–67. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

# 第 5 章

# 理論的調査研究 – プログラミング力と経験の関係，スキーマ構造，演繹以外の推論の調査

## 5.1　３つの仮説の提案

本研究で提案する内容について，理論的な仮定が正しいかどうか調査研究を行う．仮説としては以下になる．(i) を実験１，(ii) を実験２，(iii) を実験３にてそれぞれ検証を行う．

(i) プログラミング力と経験は関連する
(ii) プログラミングの知識はスキーマ構造である
(iii) 演繹以外の推論が使用される

## 5.2　プログラミング経験がプログラミング力に与える影響の調査（実験１）

### 5.2.1　実験内容の詳細

プログラミングの経験量と開発力の関連を調査した [1]．なお，本研究においては開発力とは理解したアルゴリズムをコーディングする能力である．

実験の流れは以下の通りである．

1. プログラミングの経験量のアンケート回答 (3 題)

2. 制御構文に関する一般的な知識課題回答 (3 題)
3. 経験量が必要と想定される課題 (35 題)
4. 開発力を問う課題回答 (2 題)

となる．なお，今回使用するプログラミング言語としては，対象とする被験者の学科で必須科目である C 言語を採用した．

本研究の仮説としては，以下となる．

**プログラミング力と経験は関連する**　アルゴリズムの理解と実装の得点を経験別に比較することで，経験が多いものは「理解はできて．実装もできる」，経験が少ないものは「理解はできるが，実装はできない」という状態になるかどうかを確認する，

**特定の課題は経験を反映する**　各種の経験能力が反映すると思われる課題の得点が経験別の分類で特定の傾向が見られるか確認する．傾向が見られる場合は，その課題については経験を反映する課題と見なせる．

(a)　**アンケート (計 3 題)**

これまでのプログラミングの経験等について入門科目の成績，応用科目の成績，普段プログラミングを行うか，という計 3 点を質問した．

(b)　**知識課題 (計 3 題)**

変数宣言・反復・分岐に関する基本的な知識を問う問題をそれぞれ 1 題，計 3 題出題する．問題としては，虫食いのソースコードを指定の結果となるように穴埋めを行う課題とした．

(c)　**経験課題 (計 35 題)**

経験量に関連があると考えられる 3 種の課題群を作成した．それぞれ経験量から得られる能力に関連すると考えられる課題となる．

**瞬間判断課題 (25 題)**　手続き化の視点より，2,3 行のソースコードを見せて短時間（5 秒以内）に，C 言語の処理・文法として正常か異常かどうかを判定する．

**柔軟性課題 (5 題)**　特定の制限のもと，指定された条件を満たす処理を書く．具体的には，繰り返しを要する課題で for 構文の使用禁止・使用回数指定，出力関数の使用回数制限となる．その他にも，変数の個数を制限しての総和計算問題などがある．

Table. 5.1　Scores of the algorithm understanding and implementation for each group (Do programming often group v.s. Do not group)

| | | | Understand | | Development | |
|---|---|---|---|---|---|---|
| | | n | M | (SD) | M | (SD) |
| PG | YES | 8 | 85.6 | (19.5) | 70.6 | (26.1) |
| often? | NO | 10 | 68.5 | (27.8) | 4.0 | (12.6) |

**記憶課題 (5 題)**　短時間（15 秒）でプログラミングコードを記憶する．なお，問題が進むにつれてソースコードが長くなる（前回の問題に追加）課題とした．これにより，チャンク未獲得時の負荷を大きくし，経験量の影響が出やすくなると考えられる．

(d)　**開発力課題 (計 2 題)**

初見のアルゴリズムを教授し，これを実装できるか問う課題となる．今回は本学の授業で教授を行わないソートアルゴリズムであるバケットソートを使用した．

**アルゴリズム理解課題 (1 題)**　図と文字でアルゴリズムを説明し理解させた後に，プログラミングをさせずに，理解の確認を行う．具体的には，数値が書かれた カードを用いて，教授されたアルゴリズムを使用してソーティングを行う．

**アルゴリズム実装課題 (1 題)**　教授されたアルゴリズムを実装する．本研究では，この課題に正答出来るものは，高い開発力とみなす．

### 5.2.2　結果

本実験について，被験者の募集にあたり報酬などは用意せずボランティアで参加可能なもののみが参加した．実験にあたり，高得点や低得点などの実験結果により扱いは変わらないことを説明し，得点を競うものではなく，試験問題に対して各人の自然な反応をすることを重点的に説明した．実験の参加者は 19 名であったが，1 名受験手続きを誤ったため回答が正常に行えなかった．そのため，この 1 名を除いた 18 名の被験者の回答データを解析した．被験者について全員が東海大学情報通信学部の学生であり，入門科目の単位取得済み 17 名，応用科目の単位取得済みは 13 名であった．

経験とプログラミング力の関連調査については Table. 5.1, Table. 5.2 の結果となった．n は被験者数，M は平均得点（Mean），SD は標準偏差（standard deviation）を表す．

Table. 5.1 は，普段プログラミングを行うグループと行わないグループそれぞれの「アル

Table. 5.2 Scores of the algorithm understanding and implementation for each group (Grades of introductory and advanced programming class)

| | | | Understand | | Development | |
|---|---|---|---|---|---|---|
| | | n | M | (SD) | M | (SD) |
| Intro | Low | 1 | 70.0 | (—) | 0 | (—) |
| | Mid | 9 | 81.1 | (27.1) | 18.8 | (24.5) |
| | High | 8 | 71.2 | (25.7) | 54.3 | (45.6) |
| Advanced | Low | 5 | 79.0 | (20.0) | 8.0 | (17.8) |
| | Mid | 4 | 57.5 | (25.0) | 10.0 | (20.0) |
| | High | 9 | 82.7 | (26.5) | 58.3 | (39.4) |

ゴリズム理解課題」(Understand) と「アルゴリズム実装課題」(Development) の結果である．「アルゴリズム理解課題」について，普段プログラミングを行うグループの得点と，行わないグループの得点に有意差が認められなかった（$t(16) = 1.46, p = .16$）．しかし，「アルゴリズム実装課題」(Development) は平均得点に開きがあり，0.1% 水準で有為差が認められた（$t(16) = 7.12, p < .001$）．

なお，t(16) 以降の表記は t 検定の結果で 16 は自由度を表し，7.12 は t 値，p<.001 は p 値を表す．p 値については，この数値が低いほど検定の対象が異なる可能性が高い (推定される母集団の平均値が一致しない可能性が高い)．$p < 0.05$ で検定対象が異なると判断する (優位差あり)．本研究では一般的に使用されている次の表記を採用する．p<0.05(5% 水準) で*, p<0.01(1% 水準) で**，p<0.001(0.1% 水準) で***.

プログラミング授業成績との関係ついては，Table. 5.2 の通りとなる．授業成績は，高い方から S,A,B,C, 不合格, 未履修となるものの，今回は被験者が少なく各分類に十分な人数が揃わないため，High（S,A），Mid（B,C），Low（不合格, 未履修）の 3 グループとした．

初めに，入門科目 (Intro) の成績別得点比較であるが，1 要因の分散分析を行ったところ，「アルゴリズム理解課題」（$F(2, 15) = 0.3, p = 0.73$）も「アルゴリズム実装課題」も（$F(2, 15) = 2.5, p = 0.11$），High, Mid, Low それぞれのグループ間に有為な得点差は認められなかった．次に，応用科目 (Advanced) についても同様に分散分析を行ったところ，「アルゴリズム理解課題」（$F(2, 15) = 1.4, p = 0.25$）は有為な得点差が認められなかったが，「アルゴリズム実装課題」（$F(2, 15) = 5.53, p = 0.016$）は 5% の水準で有意であった．なお，F は分散分析の結果であり，(2,15) は自由度を表し，0.3 は F 値，p=0.73 は p 値を表す．

有意な差が認められた「アルゴリズム実装課題」の応用科目成績別得点について，Tukey 法による多重比較を行い分類間の得点差を確認したところ，High-Low 間に 5% 水準で有為な得点差が認められたが，その他の，Mid-Low と High-Mid については，有為な差は認められな

Table. 5.3　Summary of the three groups that is divided with clustering method

| | Correlation | Mean |
|---|---|---|
| Group A | Low | Low |
| Group B | Middle | High |
| Group C | High | Low |

かった．

開発力を問う「アルゴリズム実装課題」の得点と「知識課題」と「経験課題」の正答傾向の関連を調査する事で，一定の開発力を身につけた者と，そうでない者が，正答できる課題の難易度の境界線を調査する．分析の手法としては，クラスタ分析を使用した．

クラスタ分析に使用した変数は 2 つあり，1 つは各問題の平均点であり，もう 1 つは各問題の得点と「アルゴリズム実装課題」の得点との相関値とした．これにより，開発力が必要な問題と，易しすぎる・難しすぎる問題のグループ分けを行う．

分析は，各得点を標準得点に変換し，ウォード法，ユークリッド距離の平方にて分析を行った．クラスタ数を 3 にして分類した結果，以下の Table. 5.3 の特徴があった．

この 3 グループに対して分散分析を行い，相関値 (Correlation)（$F(2,37) = 41.2, p < .001$）と平均点 (Mean)（$F(2,37) = 54.8, p < .001$）のどちらも 0.1% 水準で有意であり，Tukey 法による多重比較でもグループ A とグループ C の平均点以外は全ての組み合わせで，5% 水準で有意な得点差があった．

個別の各問題のグループ分けは Table. 5.4 に示す．これは，それぞれの個別の問題が，グループ A, B, C のどれに属すのかと，それぞれの問題の簡単な説明を記載した．

### 5.2.3　考察

普段のプログラミング経験は明らかな開発力への影響が見られた．しかし，アルゴリズム自体の理解力に関しては，プログラミング経験での有為な差は認められなかった．この点から見ても，一定以上の経験があるとアルゴリズムを理解して実装する事が可能であり，経験を伴う深い理解がないと，アルゴリズムを理解する事は出来るが，実装する事が難しいという事が分かる．

プログラミング科目の成績も一定の開発力への影響が見られ，応用科目の高成績グループと，低成績グループ間に差が認められたものの，経験に比べ明らかに強い影響ではない．そのため，高い成績を取得していても，普段プログラミングを行わない者は開発力が必ずしも伴わないという事が分かる．

Table. 5.4 Details of the three groups that is divided with clustering method

| Q | G | Type | Description |
|---|---|---|---|
| Task of knowledge | | | |
| 1 | A | Fill in a blank | Declaration of floating point number |
| 2 | B | Fill in a blank | Selection statement |
| 3 | B | Fill in a blank | Iterative statement |
| Task of short time descision | | | |
| 1 | C | Two choice question | A valid variable declaration |
| 2 | A | Two choice question | Valid variable declaration |
| 3 | A | Two choice question | A variable declaration with syntax error |
| 4 | A | Two choice question | Variable declarations with syntax error |
| 5 | A | Two choice question | A variable declaration with syntax error (casting) |
| 6 | A | Two choice question | A valid variable declaration (casting) |
| 7 | C | Two choice question | A valid assignment to array value |
| 8 | A | Two choice question | An assignment to an array value with syntax error |
| 9 | C | Two choice question | An assignment to an array value with syntax error |
| 10 | A | Two choice question | An assignment to a variable with syntax error |
| 11 | A | Two choice question | An assignment to an array value with syntax error |
| 12 | A | Two choice question | An assignment to a variable with syntax error |
| 13 | A | Two choice question | An assignment to a variable with syntax error |
| 14 | C | Two choice question | A valid selection statement |
| 15 | A | Two choice question | A selection statement with syntax error |
| 16 | A | Two choice question | A selection statement with syntax error |
| 17 | A | Two choice question | A valid selection statement |
| 18 | A | Two choice question | A valid iteration statement |
| 19 | A | Two choice question | A valid iteration statement |
| 20 | A | Two choice question | An iteration statement with syntax error |
| 21 | A | Two choice question | An iteration statement with syntax error |
| 22 | A | Two choice question | An iteration statement with syntax error |
| 23 | A | Two choice question | An iteration statement with syntax error |
| 24 | A | Two choice question | A valid iteration statement |
| 25 | A | Two choice question | An iteration statement with syntax error |
| Task of applied skill | | | |
| 1 | B | Implementation | Must use iteration statement |
| 2 | B | Implementation | Can not use iteration statement |
| 3 | B | Implementation | Can not use iteration statement and limit outputs |
| 4 | C | Implementation | Calculation of summation of an array with specified variables |
| 5 | C | Implementation | Calculation of odd value with specified array |
| Task of recall skill | | | |
| 1 | B | Implementation | Output in iteration statement |
| 2 | B | Implementation | Variable conversion of iteration specification in addition to the above processing |
| 3 | C | Implementation | Output once every 4 times in addition to the above processing |
| 4 | C | Implementation | Output switching with selection statement in addition to the above processing |
| 5 | C | Implementation | Output counter in addition to the above processing |

各課題の分類であるが，グループ A は開発力と相関が低いため開発力に影響の無い問題群であり，グループ B は平均点が高く相関が中程度であるため一定の理解があれば皆が正答できる問題であり，グループ C は開発力を身につけている者が正答できる問題であり，正答には一定以上の経験が必要であると考えられる．

特にグループ B は，変数操作のない単純な関数呼び出しやネストの無い for 構文の問題であり，グループ C は，使用変数が制限された総和計算や，for の中に if が入るレベルの処理の記憶課題となった．また，瞬間判断でグループ C に属したのは，配列を扱う問題と，if(1) 等の初歩的な処理には見られない問題であった．

この点を検討すると，分岐や反復の基本的な部分を理解していても経験が少ない者は，反復構文がネストし，かつ，変数操作やを余剰計算を伴うレベルの処理や，配列を扱う処理は実装が容易ではない事が分かる．この部分が，一定の経験で基本的な理解から使える知識に変える必要がある要素の１つではないかと考えられる．

なお，瞬間判断課題は，開発力との関連性が全体的に低かった．関連があった問題も，変数の定義や代入，if 文であり，しかも正常系の問題が多かった．この原因としては，文脈無しに特定の処理だけを質問した事が原因と考えられる．各処理は，文脈や目的に関連していると考えると，抽象的な１つの処理のまとまりの瞬間的な判断は，熟達者でも容易ではなく，単純な正常系の質問のみ正答できたと考えられる．

### 5.2.4　まとめ

プログラミング力の分類について，普段のプログラミング経験による分類やプログラミング科目の成績による分類で比較したところ，普段のプログラミング経験と言う分類がもっともプログラミング力を明確に表していた．そのため，プログラミング力は，普段の経験によって培われる能力が強く関連していると言うことを示す結果となった．

また，プログラミングの課題については，制限課題や記憶課題などを行ったが，これは特定の課題の種類がプログラミング力と関連しているわけではなく，課題で出題されるプログラミングコード自体が強く関連しているため，経験により様々な課題に対処できる力がつくが，回答できないプログラミングコードは他の課題でも回答できないと言う結果となった．（記憶できないコードは，柔軟性が発揮できず，柔軟性が発揮できないコードは記憶できない）

# 参考文献

[1] 菊池 智 , 濱本 和彦. プログラミングの経験量が開発力に与える影響に関する研究. 先進的学習科学と工学研究会, 73:31–35, mar 2015.

## 5.3　プログラミング力とスキーマ的な知識の構成に関する調査（実験２）

### 5.3.1　実験内容の詳細

プログラミング力が高い学習者と低い学習者の知識の特徴の比較することで，それがスキーマ的な構成であるかどうかを調査した [1].

プログラミング書き課題は，プログラミングの応用力と保有している知識を調査するのに適した対象である．しかしながら，一度の試験において複数の問題を問うことは容易ではない．なぜならば，問題を解くことに時間がかかるからである．したがって，そのような手法をとった場合は調査対象の範囲は狭まる．一方，トレース課題（コードを見て出力結果を記載）は少ない時間で回答が可能であるため，幅広い出題が可能であるため，今回の調査に適している．

従来の研究では，トレーススキルと書きスキルの関連について報告があるが [2]，これによるとトレーススキルは書きスキルの下位に属すという結果となる．トレーススキルと書きスキルの関連がある事を踏まえると一つの試験において，トレース課題と書き課題を混在させる事は可能である．本研究では，次の点を調査するため，書きとトレースの両方の課題を用いる．

1. スキーマの詳細 (プログラミング書きスキルに関連した知識)
2. スキーマと応用力との関係
3. スキーマと覚えたての脆い知識との違い

実験は３セクションに分かれている．それぞれの問題概要は Table. 5.5 に示す.

#### (a)　セクション 1: 書き課題 1

初めのセクションは書き課題になる．これは被験者の C 言語スキルを調査する目的となる．出題される要件を満たすプログラミングコードを記載する課題であり，難易度は入門程度である．この課題は 7 問あり，Q1, Q2 は条件分岐処理と変数出力である．Q3, Q4 は配列の出力である．Q5, Q6 は配列の演算である．Q7 は配列の最大値の出力である．Q2, Q4, Q6 は応用的な問題が出題される．この結果はのちの解析にて使用される．高い得点結果のグループは，高いプログラミング力を持つため，「高レベル群」とし，低い得点結果のグループは「低レベル群」とする.

Table. 5.5　Question list of the experiment.

| Stage | Type | Detail | | Number of questions |
|---|---|---|---|---|
| 1 | Writing | Output variable with selection | | 2 |
| | | Output values of an array | | 2 |
| | | Calculation values of an array | | 2 |
| | | Output maximum value of an array | | 1 |
| 2 | Reading | Normal | Output | 7 |
| | | | Calculation | 9 |
| | | | Selection | 5 |
| | | | Iteration | 11 |
| | | Syntax error | Output | 5 |
| | | | Calculation | 3 |
| | | | Selection | 2 |
| | | | Iteration | 3 |
| 3 | Writing | Output minimum value of an array | | 1 |

```
OUT-01 (Stage 2 Q1)
     printf("aaa\nbbb\n");
OUT-02 (Stage 2 Q5)
     int a = 10;
     printf("%d",a+3);
OUT-03 (Stage 2 Q7)
     printf("aaa\tbbb\t");
OUT-04 (Stage 2 Q8)
     printf("%c", 'a');
OUT-05 (Stage 2 Q9)
     printf("%.1f", 1.1);
OUT-06 (Stage 2 Q10)
     printf("%05d", 101);
OUT-07 (Stage 2 Q12)
     int arr[10];
     arr[1] = 13;
     printf("%d",arr[1]);
```

Fig. 5.1　Questions of Reading task (Output executable). Include and main function statement is abbreviated. OUT-01 is the name of the question and Q1 is the order in the experiment.

### (b)　セクション 2: トレース課題

次のセクションはトレース課題である．被験者はプログラミングコードを見てコードの結果を三択の選択肢（実行可能，文法エラー，終了しない）から選択する．そして，実行可能を選択した場合は，出力結果を記入欄に記載する．難易度はセクション 1 と同程度である．

このセクションは，書きスキルに関連した知識の調査とスキーマ的な知識の調査のために設けた．45 問出題される．8 グループに分割した様々なコードが出題される．このグループは，4 種の構造（出力，演算，分岐処理，反復処理）に 2 種の状態（出力可能，文法エラーまたは終了せず）をもたせたものとなる．詳細は Fig. 5.1–5.8 に示す．

被験者は各問題に対して，さらに四択の回答も行う．これは，このコードを見た事がない，

```
EOUT-01 (Stage 2 Q2)
     printf "abc";
EOUT-02 (Stage 2 Q3)
     printf<"abc">;
EOUT-03 (Stage 2 Q4)
     printf(66);
EOUT-04 (Stage 2 Q6)
     printf(abc);
EOUT-05 (Stage 2 Q11)
     printf("It is "THE BEST" car!!");
```

Fig. 5.2　Questions of Reading task (Output syntax error). The format is the same as Fig. 5.1.

```
CAL-01 (Stage 2 Q13)
     int a=20;
     a++;a--;a++;
     printf("%d",a);
CAL-02 (Stage 2 Q14)
     int a=21;
     printf("%d",a++);
CAL-03 (Stage 2 Q15)
     int a=11;
     a = 1;;;;;
     printf("%d",a);
CAL-04 (Stage 2 Q18)
     int a=15;
     a += 1;
     printf("%d",a);
CAL-05 (Stage 2 Q20)
     int a=17;
     a++;
     printf("%d",a);
CAL-06 (Stage 2 Q21)
     int a=18;
     a+5;
     printf("%d",a);
CAL-07 (Stage 2 Q22)
     int a=19;
     a==10;
     printf("%d",a);
CAL-08 (Stage 2 Q23)
     int a=10, b=6;
     a = b = 20;
     printf("%d, %d",a, b);
CAL-09 (Stage 2 Q24)
     int a=10, b=5;
     a = a<=b;
     printf("%d, %d",a, b);
```

Fig. 5.3　Questions of Reading task (Calculate executable). The format is the same as Fig. 5.1.

```
ECAL-01 (Stage 2 Q16)
     int a=12;
     10 = a;
     printf("%d",a);
ECAL-02 (Stage 2 Q17)
     int a=13;
     =a;
     printf("%d",a);
ECAL-03 (Stage 2 Q19)
     int a=16;
     a.a = 20;
     printf("%d",a);
```

Fig. 5.4　Questions of Reading task (Calculate syntax error). The format is the same as Fig. 5.1.

```
SEL-01 (Stage 2 Q25)
     int a=20, b=20;
     if(a==b) printf("a==b");
     if(a>=b) printf("a>=b");
     if(a<b) printf("a<b");
SEL-02 (Stage 2 Q26)
     int a=10, b=21;
     if(a!=b)
       printf("a!=b");
SEL-03 (Stage 2 Q27)
     int a=9, b=20;
     if(a>b){
       printf("a");
     } else {
       printf("b");
     }
SEL-04 (Stage 2 Q29)
     if(1){
       printf("abc");
     }
SEL-05 (Stage 2 Q30)
     int a=11, b=20;
     if(a>b)  a=30; printf("%d",a);
```

Fig. 5.5　Questions of Reading task (Selection executable). The format is the same as Fig. 5.1.

```
ESEL-01 (Stage 2 Q28)
     int a=10, b=22;
     if(a!b)
       printf("a!b");
ESEL-02 (Stage 2 Q31)
     int a=10, b=20;
     if(a>b){
       printf("a>b");
     } else {
       printf("other");
     } else if(a<b) {
       printf("a<b");
     }
```

Fig. 5.6　Questions of Reading task (Selection syntax error). The format is the same as Fig. 5.1.

見たことはあるが書いたことはない，何度か書いた事がある，多数書いた事がある，の 4 つから自己申告で経験を選択する．これは応用力を調査するために設けた問題となる．見た事がないと回答した問題であれ，どの程度正答できるかを計ることで，それを応用力とみなし計測を行う．

### (c)　セクション 3: 書き課題 2

セクション 3 は再度書き課題となる．ここは 1 問のみの出題となる．これは，セクション 1 Q7 と，Fig. 5.7 に示されるセクション 2 Q43 に類似した問題となる．この，セクション 3 Q1 は，配列の最小値を出力するコードであるが，類似問題は最大値の出力である．そのため，被験者はこの問題の前に類似の問題を見ていることになり，その記憶が参考となりそれを応用できるかが問われる．

この問題の意図としては，プログラミング学習における効果的な活動の調査となる．過去に

```
ITE-01 (Stage 2 Q32)
     int i=0;
     for(i=0;i<3; i++)
       printf("a");
ITE-02 (Stage 2 Q34)
     int i=0;
     for(i=0;i<4; i=i+2)
       printf("a");
ITE-03 (Stage 2 Q36)
     int i=0;
     for(i=0;i==5; i=i+2)
       printf("a");
ITE-04 (Stage 2 Q37)
     int i=0;
     for(i=0;i<3; i++){
       printf("%d",i);
     }
ITE-05 (Stage 2 Q38)
     int i=0;
     for(;i<3; i=i+2)
       printf("a");
ITE-06 (Stage 2 Q39)
     int i=0, a=0;
     for(i=0;a<3;i++ ){
       printf("%d",i);
       a = a+2;
     }
ITE-07 (Stage 2 Q41)
     int arr[3] = {2,5,4}, i;
     for(i=0;i<3; i++){
       printf("%d,",arr[i]);
     }
ITE-08 (Stage 2 Q42)
     int arr[3] = {1,4,3}, a=0, i;
     for(i=0;i<3; i++){
       a = arr[i];
     }
     printf("%d",a);
ITE-09 (Stage 2 Q43)
     int arr[3] = {2,5,3}, a=0, i;
     for(i=1;i<3; i++){
       if(arr[a] < arr[i]){
         a = i;
       }
     }
     printf("%d",arr[a]);
ITE-10 (Stage 2 Q44)
     #include<stdio.h>
     #define N 3
     int main(){
       int arr[N] = {4,3,5}, i;
       for(i=0;i<N;i++){
         printf("%d,",arr[i]);
       }
     }
ITE-11 (Stage 2 Q45)
     #include<stdio.h>
     #define N 3
     int main(){
       int arr[N] = {4,3,5}, i;
       for(i=0;i<N;i++){
         printf("%d,",N);
       }
     }
```

Fig. 5.7　Questions of Reading task (Iteration executable). The format is the same as Fig. 5.1 except Q44, Q45.

```
EITE-01 (Stage 2 Q33)
     int i=0;
     for(i=0;i<5; i=i-1)
       printf("a");
EITE-02 (Stage 2 Q35)
     int i=0;
     for(;;){
       printf("a");
     }
EITE-03 (Stage 2 Q40)
     int j=0;
     for(;j<3; )
       printf("a");
       j=j+1;
```

Fig. 5.8　Questions of Reading task (Iteration infinite loop). The format is the same as Fig. 5.1.

見た事があるコードを応用する事ができるかどうか問う狙いがある．初出の問題に回答できず，最後の問題に正答できたならば，問題を見ること自体も有効であり，逆の結果となればコードを見ることが有効ではない可能性が示される．

### 5.3.2　結果

本実験について，被験者の募集にあたり 1.5 時間程度の拘束となるため 1,500 円の金券を報酬として用意した．実験にあたり，高得点や低得点などの実験結果により扱いは変わらないことを説明し，得点を競うものではなく，試験問題に対して各人の自然な反応をすることを重点的に説明した．25 人の被験者が実験に参加し，全員が大学でプログラミング授業を受けた経験がある．書きとトレース課題は最大 100 点で採点され，各問題には均等に配点がなされた．

書き課題の平均得点は 50.0（SD = 29.2）であった．被験者は，書かれたコードが正しいと判断し，分岐または反復処理が適切に使用されていると見なされる場合は加点される．（例として，出力に使用される分岐処理，反復処理内で使用される分岐処理など）．誤りがあった場合は減点となる．（例として，変数定義忘れ，変数初期化忘れなど）．

トレース課題の平均得点は 50.6 であった（SD = 13.0）．文法エラーまたは処理が終わらないコードの問題では，正しい選択肢（実行可能，構文エラー，未完了）を選択した場合，その問いの満点が与えられた．実行可能コードの問題では，正しい処理の選択肢を選択した場合は得点の半分，書かれたコードによる処理結果が正しい場合はもう半分の得点が与えられる．出力結果のフォーマットに誤りが含まれている場合は，得点の 4 分の 1 が減算される．

高レベル群と低レベル群の分類のために，ウォード法を用いた階層的クラスタ分析を使用した．変数の距離計算はユークリッド距離の平方とした．この分析により，14 人の被験者が高レベル群，低レベル群には 11 名が分類された．比較には t 検定が用いられれ，多重検定の対

Table. 5.6　Score difference between High group and Low group in each code type. (Bonferroni corrected p-values)

| Type | Group | Mean(SD) | Difference |
|---|---|---|---|
| Executable | High | 61.7 (14.3) | *** |
| | Low | 36.4 (10.4) | t(23)=4.93,p<0.001 |
| Syntax error | High | 73.7 (7.3) | |
| | Low | 65.9 (9.3) | t(23)=2.36,p=0.054 |

*** p < .001

Table. 5.7　Characteristics of each variable of the clusters

| | Cluster1 | Cluster2 | Cluster3 | Cluster4 | Cluster5 | Cluster6 |
|---|---|---|---|---|---|---|
| Average | High | High | Mid | Low | Mid | Low |
| Variance | Low | Mid | High | Mid | High | Mid–Low |
| Correlation coefficient | Low | High–Mid | High | Mid | Low | Low |

応として，Bonferroni 補正が用いられた．

### (a)　コード種別による高低群の差の分析

実行可能な問題については，Bonferroni 補正後の高低両レベル群の差は（t(23)=4.93,p<0.01）で有意差があった．詳細は Table. 5.6 に示す．文法エラー問題には有意差は見られなかった．

### (b)　トレース課題に使用される問題の分類

45 あるすべての問題は，クラスター分析によって分類された．分析に使用された変数は次の三点である．

1. 平均点 (variable 1)
2. 分散 (variable 2)
3. 相関係数（トレース課題の各問題と書き課題の得点間）(variable 3)

問題を 6 のクラスターに分類し，分散分析の結果は以下の通りとなる．variable 1 の差の信頼度は 0.1% で優位 (F(5,39)=28.09,p<.001)．variable 2 の差の信頼度は 0.1% で優位 (F(5,39)=39.33,p<.001). variable 3 の差の信頼度は 0.1% で優位 (F(5,39)=19.20,p<.001).

Table 5.7 は各クラスターの特徴，そしてカテゴリの詳細は Table 5.8 に示す．

Table. 5.8　Categories of each reading task question. The text in a cell (e.g. OUT-01) is the name of a question showed in Fig. 5.1–5.8

| | Cluster1 | Cluster2 | Cluster3 | Cluster4 | Cluster5 | Cluster6 |
|---|---|---|---|---|---|---|
| Output | EOUT-04 | OUT-01<br>OUT-03<br>OUT-07<br>EOUT-01<br>EOUT-02 | OUT-05<br>OUT-06 | | OUT-02<br>OUT-04<br>EOUT-03<br>EOUT-05 | |
| Calculate | ECAL-02 | CAL-05 | CAL-01<br>CAL-04<br>ECAL03 | CAL-02<br>CAL-03<br>CAL-07 | CAL-06<br>CAL-08<br>ECAL-01 | CAL-09 |
| Selection | ESEL-01 | SEL-03 | SEL-02<br>SEL-05 | SEL-01<br>SEL-04 | ESEL-02 | |
| Iteration | | ITE-04<br>ITE-07<br>ITE-10 | ITE-01<br>ITE-02<br>ITE-06<br>ITE-08<br>ITE-09 | ITE-03<br>ITE-05<br>ITE-11<br>EITE-02<br>EITE-03 | EITE-01 | |

Table. 5.9　Score difference between High group and Low group in each experience. (Bonferroni corrected p-values)

| Type | Group | Mean(SD) | Difference |
|---|---|---|---|
| Haven't seen | High | 50.1 (19.9) | * |
| | Low | 29.7 (12.5) | t(23)=2.95,p=0.043 |
| Have seen | High | 66.0 (25.0) | |
| | Low | 50.9 (20.2) | t(22)=1.60,p=0.739 |
| Have written a few | High | 78.6 (19.0) | |
| | Low | 61.2 (19.0) | t(21)=2.15,p=0.261 |
| Have written a lot | High | 90.3 (13.9) | |
| | Low | 76.4 (27.2) | t(19)=1.56,p=0.811 |

* $p < .05$

### (c)　経験面での結果

トレース課題では，各問題に対する 4 つの選択肢を用いて，被験者に経験を質問した．（このコードを見た事がない，見たことはあるが書いたことはない，何度か書いた事がある，多数書いた事がある）．Bonferroni 補正後の結果は，経験のない（見たことがない）問題のみが高レベル群と低レベル群の間でが有意差が見られた（t(23)=2.95, p<0.05）．他の回答の問題は有意差を示さなかった．得点の詳細は Table. 5.9 に示す．

#### (d)　書き課題 2 の結果

トレース課題（セクション 2 Q43）で 9 人の被験者が正しく答えたが，そのうち 3 人は書き課題 2（セクション 3 Q1）での要件である反復処理の定数使用をしていなかった．問題の要求は，配列の要素（定義された定数）が変更されれば場合の結果も満足させること，と言う条件がある．定義命令もすでに記載されているため，定数を記載することは必須であるためこれは不十分な回答となる．

書き課題 1（セクション 1 Q7）と書き課題 2（セクション 3 Q1）の得点の間に有意差は認められなかった（t(24)=1.77,p=0.09）トレース課題の類似問題を正解したものだけを抽出した結果も同様であった．（t(8)=1.55,p=0.159）

### 5.3.3　考察

本研究での実験は，5.3.1 で述べた 3 点の研究課題を調査するために設計されたものであり，以下のサブセクションで各問題についての結果を検討する．

#### (a)　スキーマの詳細 (プログラミング書きスキルに関連した知識)

**■コードの種別による分析**　初心者のプログラマーは一定の文法知識を所有していると考える．なぜならば，実行可能なプログラミングコードとそうでないものを高レベル群と同等に区別することができるためである．

この結果は，スキーマが理解のみに基づくのではなく，繰り返しの経験に基づくと言うスキーマ理論の観点から矛盾がないことを示している．初心者プログラマーが最も頻繁に見るプログラミングコード要素が文法であり，括弧や式は，簡単なコードから難しいコードまで，同じプログラミング言語で常に変わることがないため，このような結果になったと考えられる．

実行可能コードの知識が文法エラーよりも後から能力が向上することになるが，その理由は，コード体験の反復が少ないためであると考えられる．一般的なクラスでは，説明と例題学習や一部のコードを記載して実行したのちに，講義が進行し新しい項目に移動する．したがって，スキーマは作成されず，知識は脆弱な知識のままであり続けると考えられる．

**■クラスター分析の結果**　クラスタ分析によって分類されたクラスの特徴は以下である．

**クラスタ 1**　平均点が高く，分散も低いため全員が理解できる種別の問題

**クラスタ 2**　書きスキルに関連した問題ではあるが，クラスタ 1 と同種の分類

Table. 5.10 Abstraction of cluster analysis result. A code in the cell is an example of a belonging code in each cluster. The second column is the combined result of Cluster1 and Cluster2. (The code that begin with [c1:] belongs to Cluster1 and others belongs to Cluster2)

| | Cluster1,2<br>(syntax and typical) | Cluster3<br>(advanced usage) | Cluster4<br>(atypical usage) |
|---|---|---|---|
| Syntax | c1 : printf(abc);<br>c1 : =a;<br>c1 : if(a!b)<br>printf<”a”>; | a.a = 10; | |
| Output<br>Calculate | printf(”%d”, arr[1]);<br>printf(”\n\t”);<br>a++; | printf(”%05d”, 101);<br>a++; a−−; a++;<br>a+=1; | printf(”%d”, a++);<br>a = 1;;;;;<br>a==10; |
| Selection | if(a>b){}else{} | if(a!=b)<br>Scope of the selection | if(a==b) print(”a”);<br>if(a>=b) print(”b”);<br>if(1) |
| Iteration | for(i=0;i<3;i++)<br>printf(”%d”, i);<br>for(i=0;i<N;i++)<br>printf(”%d”, arr[i]); | for(i=0;i<3;i++)<br>printf(”a”);<br>for(i=0;a<3;i++)<br>a = a+2;<br>Maximum array value | for(;;)<br>for(i=0;i==5;i=i+2)<br>for(;i¡3;i=i+2)<br>for(i=0;i<N;i++)<br>printf(”%d”, N); |

**クラスタ 3**　書きスキルに強く相関がある問題

**クラスタ 4**　書きスキルに強く相関がある問題ではあるが，クラスタ 3 よりは難易度が高い問題

**クラスタ 5, 6**　平均点が低く書きスキルとも相関が低い高難易度もしくは領域固有な問題群

各グループに関連した特徴的な問題を Table 5.10 にまとめた．クラスタ 1 は明確な文法エラーを含むもっとも難易度が低い問題群である．このような問題はそれ単体で教授対象になることはなく学習問題にも一般には含まれない．しかしながら，被験者は一定の実行可能コードの経験を積んでいるため，非自覚的に文法に関するスキーマが形成されており，正答が可能になったと考えらえれる．

クラスタ 2 の特徴は，典型的な分岐処理や反復処理の使用例であることである (例：for(i=0;i<3;i=i+1))．ただし，それらの処理のスコープに関する問題は含まれていない．この点を踏まえると，この問題まで答えられるレベルの学習者は，柔軟な知識ではなく処理内容や反復条件分も含めて全てが一体となった大きな粒度の知識になっていると考えられる．

クラスタ 3 は演算処理の代替的な処理が含まれている．このような代替処理がスキーマのスロットになっている可能性もあり，クラスタ 2 までが回答できる学習者よりは柔軟性が増した知識であると言える．トレース課題や書き課題においてコードのバリエーションは多岐にわたるため，それに対応するにあたり，このような知識は重要である．その他の特徴としては，分

岐処理や反復処理において，スコープの範囲の処理が複雑化してきたため，この部分が変数化したと考えることができる．

クラスタ 4 は特殊な分岐や反復処理が含まれている．例としては，SEL-01, SEL-04, ITE-03, ITE-05 である．平均点が低く，高レベル群の全員が回答できた問題ではない．この点については，今回の被験者のレベルでみると，これらの問題群に対応できるまでの変数化された柔軟なスキーマ構造が獲得できていないとみなすことができるため，高レベル群の被験者も学ぶ対象として適切なレベルの問題群であると見なせる．経験によるスキーマ獲得が足りずに正答が行えない場合は，この改善として多種のコード (例： for(i=0;i<10;i=i+2), for(i=0;i<10;i=i+3)) を経験することで豊かなスロットを獲得し正答が行えるようになることが期待できる．

#### (b)　スキーマと応用力との関係

高レベル群は「見たことがない」と回答したトレース問題において，高い得点を獲得した．この点の解釈としては次が考えられる．高レベル群のスキーマが単純なリストや脆弱な知識ではなく，(a) で示される通り柔軟に形成されており，これが多種の経験により培われたものと考えられる．スキーマ構造は特徴としてスロットと代替値による多様な表現があるため，この知識が「見たことがない」と回答した問題に対応できる理由になったと考えられる．

#### (c)　スキーマと覚えたての脆い知識との違い

(d) に記載した結果より，9 人のうち 3 人の被験者が定数を使用しての適切な反復処理が記載できなかった．彼らはトレース問題で正答しているにもかかわらず，記載できないという結果であり書き課題 1 と書き課題 2 の間で有意差が見られないことからも，見るだけでは書きに影響が与えられないという可能性が示された．この結果から，少なくとも反復処理の定数に関しては読む知識と書く知識は異なり，見ることだけでは書きに影響がないと思われる．

### 5.3.4　まとめ

本研究では，スキーマの詳細と，スキーマと応用力との関係を調査した．結果として，書き問題の高レベル群は高いスキルを持ち，さらにプログラミングスキーマと考えられる共通的な知識が見られた．この結果は，プログラミングの応用力が，教科書に書かれた宣言的な知識から得られたものではなく，経験により作られたスキーマから得られたという仮説を示唆している．

スキーマ的な知識を示すと言う結果は，理解の段階を示すことが明らかになった．最初のス

テップは，計算における代替の選択肢が不十分な，典型的な文法の知識である．第 2 段階は，演算における代替選択肢や，選択および反復処理の典型的なコードの知識である．その後，反復と選択処理に関する引数部分のプロパティを持つスロットが作成され，プログラマーは理解の進歩とともに応用力を向上させる．

スキーマの鍵は経験と抽象である．したがって，プログラミングの応用力を改善する理想的な教授方法は，多少異なるプログラミングコードの多数の経験であると考えられる．そのコードの違いは，正しい共通知識（例えば，基本文法）の作成，および正しいスロットおよび特性（例えば，SEL-01，SEL-04，ITE-03，および ITE-05）の作成に焦点を当てるべきである．ただし，スキーマを作成するために必要な繰り返しの数，および他の教授方法との比較は，今後の研究課題となる．

# 参考文献

[1] Satoru Kikuchi and Kazuhiko Hamamoto. The effects of a schema on applied programming skills in an introductory class. IEEJ transactions on electronics, information and systems, 136(7):995–1000, 2016.

[2] Mike Lopez, Jacqueline Whalley, Phil Robbins, and Raymond Lister. Relationships between reading, tracing and writing skills in introductory programming. In Proceedings of the Fourth International Workshop on Computing Education Research, ICER '08, pages 101–112, New York, NY, USA, 2008. ACM.

Table. 5.11　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 |

## 5.4　プログラミングにおける演繹的な力と類推的な力の調査（実験３）

### 5.4.1　実験内容の詳細

プログラミングにおいて演繹的な力以外の存在可能性を調査するため，類推的な力の調査を行った [1]．この調査にあたっては，仮説としてプログラミングコードのトレース（出力結果記載）が出来なくとも，プログラミングコードの修正が可能であるかどうかを調査した．

演繹的な力のみでコード修正を行うと仮定すると，修正を行う前に出力結果を判定する必要がある．その上で，修正箇所を検討し修正を行う．このような流れでは修正を行うためには出力結果の把握が必須となる．もしも，そのような関係が崩れた場合は，類推など演繹以外の推論もコード修正に並列的に用いられていると考えられる．

#### (a)　実験の詳細

実験は，トレース課題と修正課題という 2 つの課題から構成されている．どちらの課題もそれぞれ 10 問あり，同様な構造で，かつ入門レベルの難易度のコードを採用した．Table. 5.11 に 問題を示す．最初の課題はコードトレースである．被験者は，C 言語で書かれたコードが与えられ，各問題の出力結果を記載する．次の課題はコードの修正である．被験者に不完全なコードと望まれる処理結果が与えられる．被験者は望まれる出力結果を満たすようなコードの修正を行う．

問題は，入門レベルでの各処理の理解の深さを調べるように設計されているため，容易な

問題（例えば単純選択）から難しい（例えば，ネストされた反復）問題に徐々に変化させた．問題 1（Ot-1）はコード変数の出力である．問題 2 と問題 3（S1-1，S1-2）は簡単なコード選択処理である．問題 4 から問題 7（It-1 から It-4）は単純な反復処理である．問題 8 と問題 9（Ns-1，Ns-2）は，ネストされた反復選択となる．問題 10（Ns-3）はネストされた反復 - 反復処理に関するものである．付録には問題の詳細を記載した．両方の課題のそれぞれの問題について，トレース課題の高得点群（高トレース群）と低得点群（低トレース群）の得点を比較することによって結果を分析した．実験の前に，被験者はアンケートに回答し，初級および応用プログラミング授業の成績，およびクラス外のプログラミング経験についての回答を行った．入門授業では，選択や反復などの基本的なプログラミング文を扱う．応用クラスは，ファイル IO，関数，構造などの高度な要素をカバーする．どちらのクラスも従来の教授方法で行うため，経験を重視した教授方法ではない．

### 5.4.2 結果

本実験について，被験者の募集にあたり報酬などは用意せずボランティアで参加可能なもののみが参加した．実験にあたり，高得点や低得点などの実験結果により扱いは変わらないことを説明し，得点を競うものではなく，試験問題に対して各人の自然な反応をすることを重点的に説明した．18 名の被験者が実験を完了し，それらのすべてが大学でプログラミング授業を受けた経験があった．トレースと修正課題にそれぞれ 100 点を割り当て，各問題は均等に点数を分配した．Table. 5.12 は，各問題の配点詳細を示している．実験の目的はプログラミング概念の理解の深さを調べることであるため，プログラミング構造の理解は最も重要な側面であると考える．したがって，プログラミング構造理解が得点の半分を占める．トレース課題では，最も重要な点は出力値の数であり，修正課題で最も重要な点はコードを修正する場所とした．トレース課題の平均得点は 73.9（SD = 28.1），修正課題の平均得点は 76.6（SD = 24.7）であった．

#### (a) 被験者の分類

我々は，被験者を 2 つの異なる側面に従って 2 つのグループに分けた．第 1 の側面は，トレース課題での得点である．ウォード法による階層的クラスタ分析を使用して分類した．変数の距離測定にはユークリッド距離の平方を使用した．この分析では，11 人の被験者が高トレース群に分類され，7 人の被験者が低トレース群に分類された．第 2 の側面は，修正課題の得点である．この分析では，高修正群に 14 名，低修正群に 4 名となった．その結果を Table.

Table. 5.12　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. |

Table. 5.13　Classification result of tracing and modification task.

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

5.13 に示す.

### (b)　読み課題の高得点群と低得点群の得点比較

Fig. 5.9 and Fig. 5.10 は，高トレース群と低トレース群の平均得点を示している．Table. 5.14 は，高低トレース群のトレース課題得点を比較している．Table. 5.15 は，高低トレース群の修正課題得点を比較している．比較は Welch の t 検定を片側で使用し，多重検定の場合には Bonferroni の修正を使用した．トレース課題では，反復（It-1 から It-4）に関する 4 つの問題が，高トレース群と低トレース群との間で $p = 0.05$ のレベルで有意差を示した．選択（Ns-1, Ns-2）を伴う反復についての 2 つの問題は，$p = 0.001$ のレベルで有意差を示し，ネスト反復

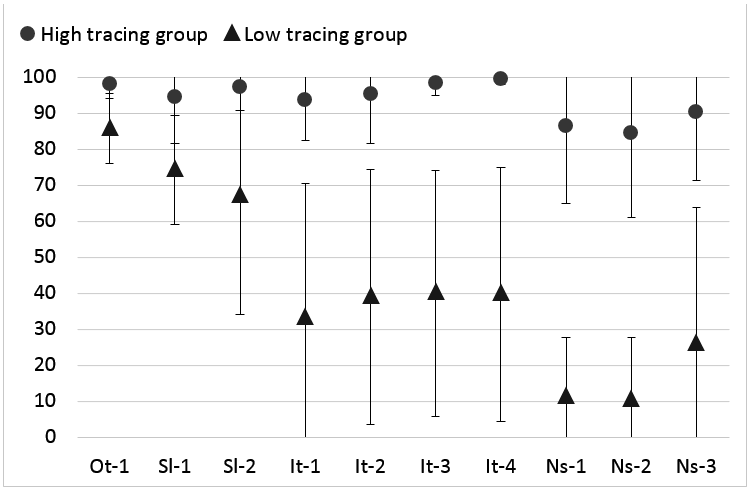


Fig. 5.9　Average score of the tracing task for high and low tracing group. X-axis is question ID, Y-axis is score of the task and error bar shows SD.

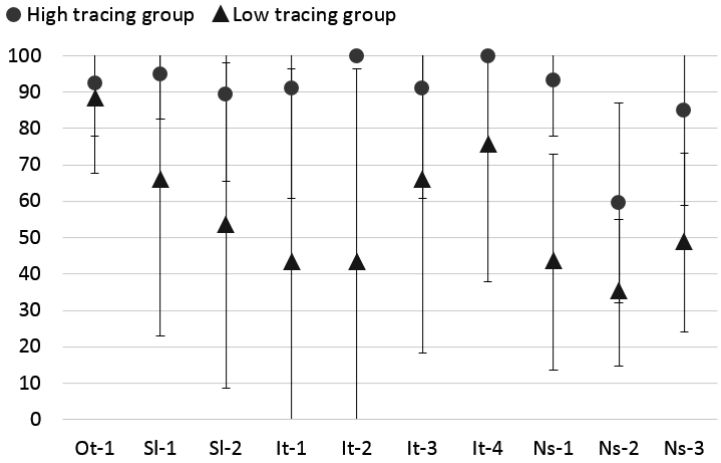


Fig. 5.10　Average score of the modification task for high and low tracing group. X-axis is question ID, Y-axis is score of the task and error bar shows SD.

反復（Ns-3）に関する問題は，p = 0.05 のレベルで有意差を示した．その他の問題では，大きな違いは見られなかった．修正課題では，ネストされた反復選択（Ns-1）に関する 1 つの問題のみが p = 0.05 のレベルで有意差を示した．他の問題は有意差がなかった．

### (c)　各課題毎のアンケート回答結果について高得点群と低得点群の比較

Table. 5.16 は，高低トレース群と高低修正群のプログラミング経験を比較した．アンケートで質問された内容は，「あなたは時々授業時間外に時間にプログラムするか？」である．「はい」と答えた 7 人の被験者はすべて，高トレース群と高修正群に属していた．「いいえ」と回

Table. 5.14　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) | t(7.3)=3.21,p=0.140 |
| | Low | 85.7 (9.8) | |
| Sl-1 | High | 94.5 (12.9) | t(11.4)=2.93,p=0.133 |
| | Low | 74.3 (15.1) | |
| Sl-2 | High | 97.3 (6.5) | t(6.3)=2.39,p=0.525 |
| | Low | 67.1 (33.0) | |
| It-1 | High | 93.7 (11.3) | * t(6.7)=4.16,p=0.046 |
| | Low | 33.1 (37.4) | |
| It-2 | High | 95.4 (13.8) | * t(7.2)=4.01,p=0.049 |
| | Low | 39.1 (35.4) | |
| It-3 | High | 98.5 (3.5) | * t(6.1)=4.52,p=0.039 |
| | Low | 40.0 (34.2) | |
| It-4 | High | 99.5 (1.5) | * t(6.0)=4.48,p=0.042 |
| | Low | 39.7 (35.3) | |
| Ns-1 | High | 86.5 (21.5) | *** t(15.2)=8.38,p<0.001 |
| | Low | 11.1 (16.5) | |
| Ns-2 | High | 84.5 (23.5) | *** t(15.5)=7.70,p<0.001 |
| | Low | 10.3 (17.3) | |
| Ns-3 | High | 90.3 (19.1) | t(8.0)=4.16,p=0.032 |
| | Low | 25.9 (38.0) | |

答した他の被験者は，高低の両群に属していた．トレーススキルに関しては，4 人の被験者が高いと評価され，7 人が低いと評価された．修正スキルに関しては，7 人が高群に，4 人が低群に属した．「いいえ」と答えた 3 名の被験者は，低トレース群であり高修正群に分類された．

Table. 5.17 は，高低トレースおよび修正群について入門プログラミングの被験者の成績を概要で示す．Table. 5.18 は応用クラスで同じ分析を示す．S は最高成績であり，C は最低成績である．N は，被験者が単位を獲得できなかったこと，またはクラスを履修しなかったことを意味する．入門プログラミングの成績では，S，A，C の成績は，課題による分類結果が異なる．3 人の被験者が低トレース群かつ高修正群に分類された．応用プログラミングの成績では，成績 C と成績 N で課題による分類結果が異なった．

### 5.4.3　考察

選択と反復処理の結果は異なる傾向を示した．反復文（It-1 から It-4）の問題で高トレース群と低トレース群の間の得点を比較すると，トレース課題では有意差が示されたが修正課題で

Table. 5.15　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) | t(9.9)=0.50,p=1.0 |
| | Low | 87.9 (20.2) | |
| Sl-1 | High | 95.0 (12.4) | t(6.7)=1.77,p=1.0 |
| | Low | 65.7 (42.8) | |
| Sl-2 | High | 89.5 (24.1) | t(8.3)=1.97,p=0.833 |
| | Low | 53.3 (44.7) | |
| It-1 | High | 90.9 (30.2) | t(8.5)=2.17,p=0.601 |
| | Low | 42.9 (53.5) | |
| It-2 | High | 100 (0.0) | t(6.0)=2.83,p=0.300 |
| | Low | 42.9 (53.5) | |
| It-3 | High | 90.9 (30.2) | t(9.1)=1.26,p=1.0 |
| | Low | 65.6 (47.2) | |
| It-4 | High | 100 (0.0) | t(6.0)=1.73,p=1.0 |
| | Low | 75.4 (37.5) | |
| Ns-1 | High | 93.2 (15.4) | * t(8.1)=4.12,p=0.033 |
| | Low | 43.3 (29.6) | |
| Ns-2 | High | 59.6 (27.5) | t(15.5)=2.20,p=0.430 |
| | Low | 34.9 (20.1) | |
| Ns-3 | High | 84.9 (26.1) | t(13.5)=2.99,p=0.101 |
| | Low | 48.6 (24.5) | |

Table. 5.16　Summary count of introductory programming class grades questionnaire answer for high and low group of each task.

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

は示されなかった．選択処理（S1-1，S1-2）に関する問題については，トレースと修正課題の両方で有意な差は見られなかった．この結果は，被験者が異なる問題について異なる推論を適用した可能性を示唆している．従来の学習方法で取得した知識は演繹に適している選択処理に関する知識と考えられる．トレース課題の結果は，上位群と下位群で差は見られなかった．つまり，低群であってもコードを読み取ることが可能であった．しかし，反復処理に関する知識は，演繹には適していないと考えられる．高群と低群のトレース得点は異なるため，被験者は帰納や類推のような他の戦略を適用した可能性がある．低群の修正得点はトレース得点よりも

Table. 5.17　Summary count of advanced 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. 5.18　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 |

悪くはなく，得点は高群の得点と同じである．これは，低群の被験者が反復処理のコードを正しく読み取ることができなかったことを意味しているが，修正課題のコードで何らかの形で修正箇所が見つかったことを意味する．読むことができないが修正することができる知識は，反復処理を書くことおよび修正することに多くの経験を得ることが容易ではない従来の教授方法で培われたものであり，被験者は反復処理の後概念を所有したとも考えられる．PBL のような経験に基づく方法を考えると，反復処理の作成や修正の経験を得ることが可能であると考えられるため，誤概念の修正が可能であると考えられる．

ネストされた反復選択（Ns-1，Ns-2）の結果は，各問題ごとに異なっていた．ネストされた反復選択に関する知識の分析のためには，Ns-1 結果が好ましい．なぜなら，Ns-2 は簡単なプログラミング文を含むだけでなく，剰余を使うので追加の知識を必要とする問題があるためで

ある．Ns-1 はトレースと修正の課題の結果に差があるため，低群はネストされた反復選択文をトレースと修正することは困難であると結論付けることができる．この理由は，反復処理のトレーススキルが不足している可能性がある．しかし，反復–反復（Ns-3）のネストされた処理に関する問題は，修正課題に違いがないことを示している．この結果は，ネストされた反復–反復処理は，ネストされた反復選択処理よりも理解しやすいことを示している．ネストされた反復–反復文（例えば，2 次元配列の初期化，ソートアルゴリズム）が頻繁に発生し，標準コードと見なされる可能性がある．しかし，これにはさらなる研究が必要であり，被験者数とコードのバリエーションを追加して調査を行うする必要があると思われる．文献 [2] では，書きスキル能力が低いことはトレーススキルの向上によって改善されることを示唆しており，読み学習を推奨している．実験の結果から，反復文のトレースよりも修正が容易なので，トレースする前に修正作業を実行することが効果的であると考えることが可能である．生徒は問題に答えるのがより快適になり，クラスのドロップアウト率が低下する．ネストされたコードのような複雑なコードの研究では，同様のコードを使った経験が有効である結果となった．アンケートでは，反復知識の誤概念解を持ち，トレースの低群に分類された．修正の高群は，入門クラスで良好な成績を達成できるが，応用クラスで好成績が取れないことを示した．

### 5.4.4　まとめ

本研究では，様々なプログラミング記述のトレースと修正スキルの関係を調査した．結果は次の通りである．(1) 被験者がトレース課題と修正課題で異なる推論を適用した可能性がある，(2) 反復処理の場合，コード修正課題がコード追跡課題より簡単である (3) 頻繁に発生するコードの経験は，コードが複雑であっても，学習プロセスを容易にする可能性がある．

結果としては，様々な教授方法を統合した方法が好ましいと考えられる．選択処理は従来手法により教授が可能であり，反復処理は経験に基づく学習が必要な可能性が示された．複雑なコードはいくつかの標準的な知識を培うような多数の経験が有効であると示された．課題としては，(1) 反復コードの修正を行う経験が反復処理のトレーススキルに影響するかどうかの調査，(2) 多数の経験を積んだ特定のコードの知識の特性調査．これが今回のネストされた反復処理と同様の結果を示すかどうか調査が必要である．

# 参考文献

[1] Satoru Kikuchi and Kazuhiko Hamamoto. Investigating the relationship between tracing skill and modification skill for different programming statements. PROCEEDINGS OF THE SCHOOL OF INFORMATION AND TELECOMMUNICATION ENGINEERING TOKAI UNIVERSITY, 9(1), September 2016.

[2] Raymond Lister, Elizabeth S. Adams, Sue Fitzgerald, William Fone, John Hamer, Morten Lindholm, Robert McCartney, Jan Erik Moström, Kate Sanders, Otto Seppälä, Beth Simon, and Lynda Thomas. A multi-national study of reading and tracing skills in novice programmers. SIGCSE Bull., 36(4):119–150, June 2004.

## 5.5　理論的調査の知見総括

実験１で経験とプログラミング力の関連が見られたためやはり経験によって得られる能力は重要であることが示された．また，経験に関する結果として実験２において明示的に行われない文法の知識が一定の経験で身につくことを踏まえると，素朴理論の影響が考えられ，低理解の学習者でも概念を正しく形成していることが示された．そして，実験３において，ネストされた反復がネストされた反復–分岐より容易である点については，コードの複雑さよりは経験によりその経験範囲の熟達化がなされた可能性が示された．

実験２でプログラミングの知識はスキーマ形式であることが確認され，特にその変数化される箇所について，一定の順番があることが示された．この流れについては，問題作成時の参考とする．また，実験２において，書き得点が高い群は，知らないと回答する読み問題も高い得点を得たことから応用力が高い結果となり，この群は同時に豊かなスキーマを獲得している事を踏まえると，スキーマが応用力に与える影響が示されたと言える．

実験３で類推的な力もプログラミングにおいて使用されている可能性が見られたため，経験の関連と同様に従来の枠組みである「説明，理解，演繹」だけではない構図が確認されたため，既存の知識，特にスキーマ的なデフォルト値の獲得は重要であることが示された．

いずれについても，期待したとおりの方向の結果を示すことになったため，当初提案したとおりの方針にて学習方法を検討する事が妥当であることが示された．

# 第 6 章

# 実践的な調査研究 – 書きと読みのそれぞれを中心とした経験型学習の効果検証

## 6.1　２つの仮説の提案

本研究で提案する内容について，実践的な仮定が正しいかどうか調査研究を行う．仮説としては以下になる．(a) を実験４，(b) を実験５にてそれぞれ検証を行う．

(a) 書きを中心とした経験型学習は理解型学習より学習効果が高い
(b) 読みを中心とした経験型学習は理解型学習より学習効果が高い

## 6.2　書きを中心とした経験型学習と理解型学習の学習効果比較検証（実験４）

### 6.2.1　実験内容の詳細

次の２つの異なる特徴を持ち合わせた学習方法の比較を行った．

1. 試行錯誤での書きを中心とした経験型学習（書き学習）
   - 問題に取り掛かれる範囲の説明 (用語の説明)
   - サンプルコードの出力結果確認
   - コード改造課題（望まれる出力結果を試行錯誤で求める）

2. 説明と実行結果の理解を中心とした理解型学習（読み学習）
   - 教科書通りの説明
   - サンプルコードの出力結果確認

### (a) 実験の概要

実験は 2 つのセクションで構成されている．最初のセクションは学習である．被験者は 2 つのグループに分けられた後，各グループごとに異なる学習教材が提供される（書き学習と読み学習）．教材の違いは，適用される学習方法である．次のセクションは試験のセクションである．学習した内容に関する問題に回答する．試験は時間を空けて 2 度実施される．学習直後の試験（学習後試験）と，学習の 3 週間後の試験（事後試験）である．学習と試験の両方のセクションはいずれもコードの実行が必要になるため，コンピュータ上の今回の実験のために用意されたシステムで行われる．

実験科目はコンピュータサイエンス学科の学生であるため，この実験での学習のテーマは，彼らが学習する内容から離れた内容とする．本実験は，事後試験があるため，学習内容と重複する内容が試験に含まれる場合は，点数に与える影響が実験であるのか，授業の内容であるかの分離が困難になるためである．被験者は主に手続き型のプログラミングで学習する．したがって，異なるプログラミングパラダイムがこの実験に適している．今回は有名な関数型プログラミング言語の 1 つであるプログラミング言語 Haskell が試験に使用された．

### (b) 学習セクション

各グループの実験教科書は，同じプログラミング教科書から作成され，提供されるプログラミングコードの数は同じである．文献 [1] が実験で使用した教科書である．これは，Haskell プログラミングの入門レベルの内容となる．被験者の主言語は日本語であるため，実際には文献 [1] の日本語版が使用された．

文献 [1] の利用された部分は，始めから第 1 章の終わりまでとなる．すべての文章とプログラミングコードは，意味は変更されずに使用された．

試験の実行を円滑にする理由で一部のプログラミングコードのスタイルは変更された．文献 [1] では，すべてのコードが”ghci”コマンドによる対話モードで実行される．しかしそれは Web ベースの命令には適していない．したがって，すべてのコードは，”runghc”コマンドによってインタプリタ・モードでの実行に変更した．変更はコードのスタイルのみであり，実行のための ”main = do”と 画面出力のための”print” であり，これらの変更は元のコードの意味を

変更するものではない.

**■読み学習** 読み学習は，文と絵を用いて定義と意味を理解する手法である．説明の途中では，プログラミング例が記述されており，被験者はコードを実行して処理結果を確認する必要がある.

コードを実行するために，実行ボタンがプログラミングコードの下に設定される．ボタンは，元の教科書で実行が必要なすべての部分に表示される.

**■書き学習** 書き学習は，プログラミングコードと説明文（名称の説明）と望まれる出力結果のみ表示される．説明文の内容は，各概念や名称である．したがって，被験者は，望ましい結果を満足する正しいコードを見つけるために試行錯誤を行うことになる.

プログラミングコードは読み学習と同一であるが，各コードの好ましい結果は，文献 [1] の結果から変更されている．学習者は最初に元のコードを実行し，その結果を確認してから修正を行う．出力結果が好ましい結果を満たすまで，コードを修正して再度実行を行う．文法エラーを含むコードの学習については，出力結果とエラーメッセージのみが表示される学習方法とした．なぜならば，これらのコードを名称のみの説明で修正することは容易ではないため，文法エラーのコード学習に限り，修正を行わないでコードを実行して見るだけの読み学習と同一の内容となる.

この学習方法は，言語の特徴を学習する場所がないため，文献 [1] の紹介ページが学習の最後に示される．この箇所は読み学習となるが，この手法を取らない限りは，のちの宣言的知識の試験に回答することが困難になるためである．この内容の提示が学習後に示される理由は，書きにより取得された学習知識と読みの内容を結合させる状況モデル理論を適用するためである [2].

2 つの条件の差を最小にするために，書き学習と同様に読み学習群にも学習終了の最後に紹介ページが提示される．理由としては，学習の最後は試験の開始と近いため，紹介ページの提示タイミングが群間で差がある場合，片方のグループが有利になると考えられるためである．紹介ページの出現数は 2 つのグループで異なるが，それぞれの学習方法の特性が異なるため影響は限られると考えられる.

### (c) 試験セクション

試験のセクションは 3 つの部分で構成される．最初の部分は 4 つの選択肢を持つ選択的な問題で 15 の問題がある．言語の意味や定義を尋ねるとともに，文法の特徴についても問題が

なされる．結果は取得された宣言的な知識について分析される．

2 番目の部分は，プログラミングコードのトレースである．被験者は最初に，2 つのオプション（実行可能，文法エラー）からプロセスの結果を選択する．次に，実行可能を選択した場合に出力結果を書き込む．同時に，経験の質問に回答する．これは 4 つの選択肢から回答する（同じコードを見たことがある，ほとんど同じコードを見たことがあり違いは値または変数の文字のみ，類似のコードを見たことがある．見たことがない）トレースは 20 問ある．結果は，コードの読解スキルとコード記憶の特徴について分析される．

最後の部分は書き課題である．3 つの問題がある．初めの 2 つの問題はコード修正である．被験者はサンプルプログラミングコードおよび望ましい出力結果をが提示される．コード修正とコードを実行を繰り返して，望ましい結果と同じ出力結果を目指す．最後の問題はすべてのコードを記載する問題となる．コードの条件と望ましい出力結果のみが被験者に示され，コードを記載して，またコードを実行し回答を目指す．

2 回目の試験は事後試験となる．これは最初のテストの 3 週間後に行われる．2 つの試験（学習後試験，事後試験）の問題はほぼ同一である．違いは，トレース問題と書き問題の望ましい出力結果となる．示されるコードと問題は変更されない．もう一点の違いは，トレース問題に 5 つの新しい問題が追加される点である．トレース問題の数は 25 に変更される．問題の詳細は付録に示す．

### 6.2.2 結果

本実験について，被験者の募集にあたり 2 時間程度の拘束となるため 2,000 円の金券を報酬として用意した．実験にあたり，高得点や低得点などの実験結果により扱いは変わらないことを説明し，得点を競うものではなく，試験問題に対して各人の自然な反応をすることを重点的に説明した．試験は 12 人の被験者によって行われ，そのすべては Haskell の学習経験はない．3 人は言語の名前だけを知っていた．学習と事後試験の間に Haskell を学習した被験者はいないため，実験以外の結果に影響を与える要因はない．年齢は 18 歳から 23 歳までであった．学習グループの分類は基本的にランダム化されているが，グループ間のプログラミング経験を最小限に抑えるために各グループの年齢の合計をできるだけ同一にするように分類した．

読み学習の学習時間の平均は 64.1（SD 8.6）分，書き学習は 56.7（SD 14.7）分である．各問題の配点は同一である．各試験パートの最大得点は正規化し 100 点とした．各問題の得点配分の詳細を Table. 6.1 に示す．

Table. 6.1 Detail of point allocation for each section

| Part | Score | Description |
|---|---|---|
| Selection | 100% | Subject get 100% of point if the selected answer is correct. |
| Tracing | 50,100% | Subjects get point if the selected process result is correct (executable, syntax error). In case the code is executable get 50%, and get 100% in case of syntax error code. |
| | 50% | Written output result is correct then subject get 50% more. |
| Write | 100% | If the answer code has correct structure, subject get 100% of point, then subtracted 10% of point if the output and variable value is not correct, subtracted points if there is a syntax error (20% for each), and subtracted 50% of point if the subjects not follow the code requirements (i.e. Modified wrong part of a code). |

Table. 6.2 Result of selection questions. Rd means the result of reading leaning method. Wr means the result of writing leaning method. Diff means the difference. (Bonferroni corrected p-values)

| | Mean (SD) | | Diff |
|---|---|---|---|
| | After study | Post | |
| Rd | 63.3 (16.2) | 47.8 (21.3) | ** t(5)=7.0,p=0.0018 |
| Wr | 63.3 (12.5) | 43.3 (17.3) | * t(5)=4.4,p=0.014 |
| Diff | t(9.4)=0,p=1.0 | t(9.6)=0.4,p=1.0 | |

### (a) 試験の得点

試験得点は，両側の Welch の t 検定を用いて 2 つの群間で比較した．また，学習後と事後試験との間で，対応のある t 検定で比較した．多重検定を考慮し，Bonferroni 補正を使用した．

選択問題の結果を Table. 6.2 に示す．学習方法間の得点の差は，学習後および事後試験の両方で示されなかった．学習後と事後試験の得点の間には，両方の学習方法において $p = 0.01$ および $p = 0.05$ のレベルで有意に異なった．

トレース問題の結果は，Table. 6.3 に示す．$p = 0.05$ のレベルで，読み学習のみにおいて，学習直後と事後試験に有意差が認められた．

書き問題の結果を Table. 6.4 に示す. いずれの学習方法も有意差が見られなかった．

コードタイプ（実行可能，文法エラー）によって結果が異なる可能性がある．初心者には文法は難しい点の一つであり [3]，トレース問題は異なる観点からも分析することが好ましい．Table. 6.5 は，実行可能（文法エラーなし）のコードのみをトレースした結果を示し，Table. 6.6 は，文法エラーの 8 つのコードの結果を示したものである．文法エラー条件の下では，読み学習のみが学習後と事後試験とで有意差を示した．

Table. 6.3 Result of trace questions. Format is same as Table 6.2

| | Mean (SD) | | Diff |
|---|---|---|---|
| | After study | Post | |
| Rd | 66.7 (9.7) | 55.0 (10.2) | * t(5)=3.3,p=0.045 |
| Wr | 53.8 (6.8) | 43.8 (5.4) | . t(5)=2.8,p=0.07 |
| Diff | . t(9.0)=2.7,p=0.052 | . t(7.6)=2.4,p=0.09 | |

Table. 6.4 Result of write questions. Format is same as Table 6.2

| | Mean (SD) | | Diff |
|---|---|---|---|
| | After study | Post | |
| Rd | 76.1 (22.6) | 74.4 (29.7) | t(5)=0.2,p=1 |
| Wr | 63.3 (34.2) | 68.3 (31.4) | t(5)=0.5,p=1 |
| Diff | t(8.7)=0.8,p=0.93 | t(10.0)=0.3,p=1 | |

Table. 6.5 Result of only executable trace questions. Format is same as Table 6.2

| | Mean (SD) | | Diff |
|---|---|---|---|
| | After study | Post | |
| Rd | 79.2 (9.1) | 69.4 (7.8) | t(5)=2.0,p=0.19 |
| Wr | 68.8 (12.6) | 54.9 (19.8) | . t(5)=2.7,p=0.08 |
| Diff | t(9.1)=1.6,p=0.27 | t(6.5)=1.7,p=0.28 | |

Table. 6.6 Result of only syntax error trace questions. Format is same as Table 6.2

| | Mean (SD) | | Diff |
|---|---|---|---|
| | After study | Post | |
| Rd | 47.9 (14.6) | 29.2 (15.1) | * t(5)=4.0,p=0.014 |
| Wr | 31.3 (27.1) | 20.8 (24.6) | t(5)=1.1,p=0.68 |
| Diff | t(7.7)=1.3,p=0.44 | t(8.3)=0.7,p=1 | |

### (b) トレース課題の経験に関するアンケート回答分析

トレースコードの経験に関する回答の結果を Table. 6.7 に示す．この結果についても，実行可能コードと文法エラーコードのそれぞれでも分析する．Table. 6.5, 6.6.

Table. 6.7 Selected number of experience answer in tracing questions. A-D means the answer of experience. A is ”Have seen same code”, B is ”Have seen, only the value is difference”, C is ”Have seen similar codes” and D is ”Haven’t seen before”. (Bonferroni corrected p-values)

| | | A | B | C | D | Diff |
|---|---|---|---|---|---|---|
| After study test (20 questions) | Rd | 20 | 56 | 38 | 6 | p = 0.21 |
| | Wr | 22 | 38 | 53 | 7 | |
| Post test (20 questions) | Rd | 2 | 7 | 104 | 7 | *** p < 0.001 |
| | Wr | 11 | 21 | 60 | 28 | |
| After study test (12 executable) | Rd | 13 | 37 | 19 | 3 | p = 0.29 |
| | Wr | 13 | 25 | 31 | 3 | |
| Post test (12 executable) | Rd | 1 | 5 | 65 | 1 | *** p < 0.001 |
| | Wr | 6 | 13 | 33 | 20 | |
| After study test (8 syntax error) | Rd | 7 | 19 | 19 | 3 | p = 1 |
| | Wr | 9 | 13 | 22 | 4 | |
| Post test (8 syntax error) | Rd | 1 | 2 | 39 | 6 | . p = 0.07 |
| | Wr | 5 | 8 | 27 | 8 | |
| Post test (5 added) | Rd | 0 | 2 | 20 | 8 | p = 0.47 |
| | Wr | 0 | 4 | 15 | 11 | |

Table. 6.8 Selected number of experience answer only the executable code in tracing questions. A-D means the same as Table. 6.7. QA-QC means the type of question. QA is the same code as in the study section. QB is almost same code, only the value is different. QC is the arranged code from the study section.

| | | after study | | | | post | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | A | B | C | D | A | B | C | D |
| Rd | QA | 10 | 12 | 8 | 0 | 1 | 1 | 28 | 0 |
| | QB | 3 | 15 | 5 | 1 | 0 | 3 | 21 | 0 |
| | QC | 0 | 10 | 6 | 2 | 0 | 1 | 16 | 1 |
| Wr | QA | 8 | 9 | 11 | 2 | 3 | 3 | 16 | 8 |
| | QB | 5 | 10 | 9 | 0 | 3 | 7 | 11 | 3 |
| | QC | 0 | 6 | 11 | 1 | 0 | 3 | 6 | 9 |

フィッシャーの正確確率検定の両側検定による学習方法の比較を行った．多重検定の対応として，Bonferroni 補正が用いられた．学習後試験で有意差はなかったが，事後試験では，全 20 問の解析と 12 問の実行可能な問題の条件の下で，試験後 p=0.001 のレベルで有意な差を示した．事後試験で 5 つの追加問題には，有意差は認められなかった．各条件の選択された解答数の詳細を Table. 6.8 と Table. 6.9 に示す．

### 6.2.3 考察

2 つの学習方法は一定の特徴を示した．それぞれの問題について考察を行う．

Table. 6.9 Selected number of experience answer only the syntax error code in tracing questions. Format is same as Table. 6.8

| | | after study | | | | post | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | A | B | C | D | A | B | C | D |
| Rd | QA | 4 | 4 | 4 | 0 | 0 | 0 | 9 | 3 |
| | QB | 2 | 12 | 8 | 2 | 1 | 1 | 19 | 3 |
| | QC | 1 | 3 | 7 | 1 | 0 | 1 | 11 | 0 |
| Wr | QA | 3 | 3 | 5 | 1 | 0 | 0 | 9 | 3 |
| | QB | 4 | 7 | 11 | 2 | 3 | 2 | 15 | 4 |
| | QC | 2 | 3 | 6 | 1 | 2 | 6 | 3 | 1 |

### (a) 選択課題

選択問題では，学習法方法間に違いはないが，学習後と事後試験の得点は，どちらの方法においても有意に異なる．これは，選択問題に答えるために必要な知識は宣言的な知識に過ぎないため，選択問題の答えの主要な要素である名称などは時間とともに忘却されるということが示された．

### (b) トレース課題

トレース問題は顕著な違いを示した．すべての問題を分析すると，2 つの学習方法の間に有意な傾向が学習後と事後試験の両方で示されるので，読み学習は有効であると考えられる．

さらに，経過時間に伴う分析の結果，実行可能なコード条件でのみ学習後と事後試験の差が見られた．読み学習は有意差が見られ，書き学習は優位傾向として示された．

これらの結果は，初心者のためのコードトレース学習については，同一条件であれば少なくとも読み学習が効率的または同じであることを示している．知識，特に文法エラーは，読み学習の条件の下で数週間で容易に忘れるという結果が示された．この結果は文献 Ref.[4] の結果と一致し，身についている文法知識は経験により得られたものと考えられる．

### (c) 書き課題

書き問題はすべての条件で違いが見られなかった．これは，問題でコードの実行を可能にしたため，知識の必要はなくコード変更スキルのみで対応できたためと考えられる．時間とともに点数が落ちるわけではなかったため，コード修正のための要件スキルは，教科書に書かれている知識に強く結びついてはおらず，特定の言語が結びつかない一般的な知識として存在する可能性が示された．

### (d) コードの経験に関するアンケート

コード経験の答えの結果もまた顕著な結果である．被験者が正しいコードを理解しているかどうかにかかわらず，学習後の両方の学習方法で経験の印象は同じであった．事後試験の結果が異なる．これは異なる学習方法が異なる記憶を作り出すことを意味すると考えられる．結果は，5 つの追加問題条件では違いがなく，両方のグループに偏りがないことを示しているので，各方法からの作成された記憶の特性を反映するものと仮定される．

事後試験では，読み学習の特徴が曖昧な記憶とみなせる．これは，「類似のコードを見た」という回答に収束するためである．これは通常の反応とみなすことが可能である．学習後 3 週間後に曖昧な想起があることを意味するが，記憶は完全に消去されず，すべてのコードは類似したコードとして記憶される．対照的に，書き学習は他の傾向を示した．この学習は読み学習の結果のような収束が強く示されない．「コードを見た」と「コードを見たことがない」という数は，理解の方法よりも大きい結果となる．この結果は，試行錯誤の経験によって，特定のプログラミングコードの記憶を形成する可能性を示している．覚えているという結果と忘れている結果が共存する理由は，修正の経緯が被験者間で異なるため，被験者間で記憶がが異なるように作成されていると考えることができる．（書きグループのための学習に必要な要素は，コード自体ではなく出力結果であった）．

1 時間の実験では，書き学習は読み学習より良い結果得点を示さなかったが，書き学習の特徴はプログラマーにメリットがあると考える．過去のコード経験の特徴は，必要に応じて想起されることが問題解決の重要なスキルであり，応用力にも重要である．書き学習は人により異なるものの，忘れにくい特定のコード記憶を形成する，と言う特徴を示している．文献 [5] や [6] のような手法で十分な時間や経験を適用した場合，応用スキルや高度なスキルを向上させる効果があると考えられる．

## 6.2.4 まとめ

本研究では，読み学習と書き学習の間の違いを調査した．その結果，読み学習は Haskell 言語の初心者のためのコードトレースを比較的効率的に，学習することが可能である．文法エラーコードの知識は数週間で忘れられることが示された．

学習方法間で記憶が異なる特徴で形成され，読み学習の記憶は曖昧になる特徴が示された．一方，書き学習の特徴は，忘れにくく，いくつかの問題に特化した特定の記憶を形成することが示された．コード修正の試行錯誤の経験によって作成された記憶が，いくつかのコードに対

して特定の記憶を形成する可能性が示された．なお，書きグループは学習中の経験するコードが異なるため，被験者間での記憶の作成は同じではない．

プログラミング学習の入門レベルの学習において．いずれの学習方法の特徴も，アルゴリズム学習や複雑なコードのような他の難易度の場合に本実験とは異なる特徴を示す可能性はある．そのような，他の難易度のコード，特に複雑なコードの調査は将来の課題とする．

# 参考文献

[1] Miran Lipovaca. Learn You a Haskell for Great Good!: A Beginner's Guide. No Starch Press, 2011.

[2] Walter Kintsch. Comprehension: A Paradigm for Cognition. Cambridge University Press, 1998.

[3] Arto Vihavainen, Juha Helminen, and Petri Ihantola. How novices tackle their first lines of code in an ide: Analysis of programming session traces. In Proceedings of the 14th Koli Calling International Conference on Computing Education Research, Koli Calling '14, pages 109–116, New York, NY, USA, 2014. ACM.

[4] Satoru Kikuchi and Kazuhiko Hamamoto. The effects of a schema on applied programming skills in an introductory class. IEEJ transactions on electronics, information and systems, 136(7):995–1000, 2016.

[5] Jos Mara Rodrguez Corral, Antn Civit Balcells, Arturo Morgado Estvez, Gabriel Jimnez Moreno, and Mara Jos Ferreiro Ramos. A game-based approach to the teaching of object-oriented programming languages. Computers & Education, 73:83 – 92, 2014.

[6] Jackie O'Kelly and J. Paul Gibson. Robocode & problem-based learning: A non-prescriptive approach to teaching programming. SIGCSE Bull., 38(3):217–221, June 2006.

## 6.3 読みを中心とした経験型学習と理解型学習の学習効果比較検証（実験５）

### 6.3.1 実験内容の詳細

次の 2 つの異なる特徴を持ち合わせた学習方法の比較を行った.

1. 出力結果選択形式の連続出題による読み中心の経験型学習（読み学習）
   - 出力結果選択 (3 択)
   - 連続出題
   - 説明の省略
   - 典型コードのみ（分岐処理の > のみや，反復処理の ++ のみ）
   - 学習の最後ににその他の種類のコードを紹介
   - 誤回答明確化
   - 誤回答時の正答表示
2. 説明と Visualize ツールと書き学習を組み合わせた理解型学習（Visualize 学習）
   - 教科書通りの説明
   - Visualize で処理確認（デバッガ形式のツール）
   - 書き課題

読み学習と，Visualize 学習と，統制群の 3 条件で比較検討を行った．実験手順を Fig. 6.1 に示す．被験者は，試験と学習との間に 5 分間の休憩を取った．学習前試験の合計得点を均等になるよう考慮し 3 つのグループに分類した．試験と学習の両方の活動は，この実験のためだけに開発された Web ベースのシステムで実行された．したがって，被験者はこの実験に参加するためにコンピュータとウェブブラウザのみを使用した.

#### (a) 試験セクション

試験は 4 つの課題で構成されている．第 1 のものは「記憶課題」であり，被験者が 3 分間プログラミングコードを記憶し，その後，2 分以内に覚えている限りコードを書き出す．文献 [1] の実験設計を参考にして，この作業を試験で 2 回繰り返す．第 2 は，被験者がプログラミングコードを読み取り，その処理結果を回答入力フィールドに入力するトレース課題である．この課題には 8 つの問題がある．3 つ目はコードの並び替え課題（並び替え課題）で，被験者

はランダムに並べ替えられたプログラミングコードを読み取り，正しい順序に並び替える．さらに，被験者はコードに含まれる未使用の行をすべて削除しなければならない．最後の課題は「選択課題」である．これには多肢選択問題となる．被験者はプログラミングコードを見て，4 つの選択肢から出力結果を選択する．この課題には 13 の問題がある．

記憶課題のコードは，Soloway の rainfall problem[2] が利用される．トレースと選択課題のコードは，選択処理，反復処理，配列の集計，ネストされた反復処理，およびバブルソートである．並び替え課題のコードは，反復処理，反復処理と演算，およびネストされた反復処理である．事前と事後の問題は同じスタイルであるが，プログラミングコードはお互いに少し異なる．記憶課題のコードのみが全ての試験で同じものを使用した．

### (b) 学習セクション

学習対象は，演算（配列を含む），選択処理（配列を含む），反復処理，配列を含む反復処理，配列の最大値または最小値，およびバブルソートの 6 つのセクションで構成される．

読み学習群は，複数の選択肢のある問題（3 つの選択肢）に回答する．それぞれの問題に対して 2 つのサンプルコードが示される．3 つの問題タイプが表示される（コードを読み取り結果を選択．虫食いコードと結果を読み取りコードを選択．結果を読み取り，コードを選択）．各問題タイプは，各セクションに 3 つの問題がある（各セクションに合計 9 つの問題がある）．3 つの問題タイプのコードは類似している．これらのタイプのコードは出題範囲が限定されるため，このグループでは典型的なコードのみが使用される．（例：出力関数は 1 つの変数のみを使用し，改行は常に最後に置かなければならず，更新の式では増分が使用され，選択条件には「>」だけが使用される）読み学習はセクション 7 が存在しており，これは典型以外のコードスタイルを紹介し，回答が必要な形式ではない．被験者はコードを読むだけである．なお，

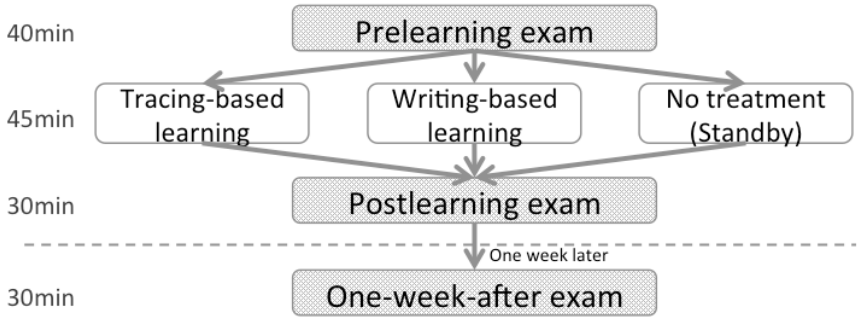


Fig. 6.1 A procedure of the experiment.

被験者が推奨される回答時間を超えた場合に警告が表示される仕様となる．

Visualize 学習群は，教科書を読む学習，Visualize 学習，および書き学習の 3 つの部分からなる学習教材を使用する．教科書は文献 [3] の日本語版で，変数，出力，選択，反復，および配列の説明に使用される．Visualize 環境は，文献 [4] の C 言語環境である [5] のシステムが使用される．Visualize には 2 つのコードが示されており，各セクションのコードは読み学習グループのように範囲に限定さない（例：出力メソッドは変数の数が少なく，改行数が少なく，インクリメントとデクリメントが 2 つ以上更新の式で使用され，選択条件には「等しい」,「より大きい」，および「より小さい」が使用される）書き学習には，1 つの問題があり，各問題の要件を満たすコードを実行する必要がある．ボタンはコンパイル結果（すなわち，コンパイルエラーまたは処理結果のメッセージ）を出力し，被験者は正しい出力を得なければならない．回答ボタンを押すと，正しいコードが被験者に表示される．読み学習と同様に推奨される回答時間を超えた場合に警告が表示される．

統制群は学習教材は提供されず，学習セクションでは自由にすることが可能である．

### 6.3.2 結果

本実験について，被験者の募集にあたり 2.5 時間程度の拘束となるため 2,500 円の金券を報酬として用意した．また，その他に報酬として実験後に希望者にはスマートフォンアプリケーションの作成について実技を交えて説明する実践的な講義を行った．実験にあたり，高得点や低得点などの実験結果により扱いは変わらないことを説明し，得点を競うものではなく，試験問題に対して各人の自然な反応をすることを重点的に説明した．

実験には 39 名の被験者が参加した．すべての被験者が一度の実験に全員出席できないため，都合 4 回実行された．2 人の被験者は 1 週間後試験に参加できなかった．したがって，37 人の被験者が実験を完了した．全 39 人の被験者が大学でプログラミングコースを受講した経験がある．39 人の被験者のうち 20 人が大学生 1 年目であり，10 人は 2 年生，7 人は 3 年生であり，2 人は 4 年生であった．

配点は，トレース，並べ替え，および選択課題の各問題で同一とした．したがって，これらの課題の最大得点は問題数と同じである．記憶課題の配点については次のセクションで説明を行う．

アンケートの回答から，ほとんどの被験者が学習直後試験と 1 週間後試験の間でプログラミングを勉強したことを明らかにした．学習グループ毎の試験と試験の間の授業時間 (90 分)

Table. 6.10 Scores of the tracing task

| Learning method | Num of subjects | Mean (SD) Prelearning | Postlearning | One-week-after |
|---|---|---|---|---|
| Reading | 13 | 2.15 (2.32) | 4.00 (2.63) | 3.77 (2.81) |
| Visualize | 13 | 2.23 (2.58) | 3.92 (2.64) | 3.77 (2.81) |
| Standby | 11 | 2.18 (2.21) | 3.27 (2.53) | 3.45 (2.35) |

Table. 6.11 Result of paired *t* test of the tracing task. (Bonferroni corrected p-values)

| Comparison | Learning method | Difference (p-value) All (No.1-8) | No.1-4,8 | No.5-7 |
|---|---|---|---|---|
| Pre VS Post | Reading | ** (0.008) | * (0.032) | * (0.024) |
| | Visualize | * (0.033) | (0.102) | (0.131) |
| | Standby | (0.122) | (0.311) | (0.206) |
| Pre VS A-week-after | Reading | ** (0.009) | * (0.032) | (0.066) |
| | Visualize | * (0.042) | (0.069) | (0.156) |
| | Standby | * (0.033) | (0.064) | (0.332) |

の平均 (標準偏差) は次の通りである．読み群 1.69（1.07），Visualize 群 1.69（0.99），統制群 2.27（1.48）となった．課題または自己学習の時間（分）の平均（標準偏差）は，読み群 22.69（32.74），Visualize 群 15.85（23.8），統制群 53.64（75.83）である．各群の研究時間を評価するために ANOVA を使用して分析した結果，結果は，F（2,34）= 1.897，p = 0.166 と群間で差は見られなかった．

### (a) 試験セクション

学習方法の特徴は，特定の課題でのみ示される可能性がある．なぜならば，各課題には，その問題に答えるために異なるスキルが必要だからである．したがって，各課題の得点を学習方法毎に分析してその違いを示す．比較は対応のある t 検定を使用し，多重検定の対応として Bonferroni 補正を使用した．

**■トレース課題** トレース課題の得点は Table. 6.10 差は Table. 6.11 に示す．すべての問題の平均点数（1〜8）は，両方の学習群で事前試験と学習後試験，学習後試験と 1 週間後試験で全て有意に差が見られた．Visualize 群の 5 名は学習活動を完了ができなかった．学習できない項目についての試験は学習できたものより不利な条件となるため，それを考慮に入れて，学習活動の後半の問題に関連する低い（1〜4,8）または高い（5〜7）の分類した結果で分析した．読み学習群のみがこれらの条件下で有意差を示した．

Table. 6.12　Scores of the sorting task

| Learning method | Num of subjects | Mean (SD) Prelearning | Postlearning | One-week-after |
|---|---|---|---|---|
| Reading | 13 | 1.31 (1.14) | 1.77 (1.37) | 1.92 (1.33) |
| Visualize | 13 | 1.00 (1.36) | 1.85 (1.35) | 1.77 (0.89) |
| Standby | 11 | 0.91 (1.00) | 2.00 (0.74) | 2.55 (0.99) |

Table. 6.13　Result of paired *t* test of the sorting task. (Bonferroni corrected p-values)

| Comparison | Learning method | Difference (p-value) All (No.1-4) | No.1-3 | No.4 |
|---|---|---|---|---|
| Pre VS Post | Reading | * (0.011) | (0.059) | (0.248) |
| | Visualize | ** (0.008) | * (0.033) | (0.248) |
| | Standby | ** (0.009) | ** (0.007) | (0.285) |
| Pre VS A-week-after | Reading | ** (0.001) | * (0.027) | (0.123) |
| | Visualize | * (0.039) | (0.156) | (0.248) |
| | Standby | *** (<.001) | *** (<.001) | * (0.024) |

**■コード並び替え課題**　Table. 6.12 はコード並び替え課題の得点を示し，Table. 6.13 はその差を示す．この課題のすべての問題（1〜4）の平均得点は，すべての条件において有意に異なっていた．トレース課題と同じ理由で，2 つの追加条件を適用した結果を分析した．学習教材の最後の問題に対する関連性の低い問題（1〜3）に関しては，Visualize 群と統制群は有意に差が見られた．関連性の高い条件（4）の下では，統制群のみに差が見られた．

**■選択課題**　ランダムに選択された回答に起因する可能性が高いノイズの影響を受けやすいため，選択課題の結果が学習グループ間の違いを分析するには不適切であると思われる．読み学習群における 1 人の被験者および Visualize 群における 1 人の被験者の得点は，学習後 4 点減少した．この減少は，ランダムに選択された回答をの影響と考えられる．したがって，本研究では本課題の結果を使用しない．

**■記憶課題**　行ごとの得点の平均得点は，Table. 6.14 に示す．厳密な採点では，入力テキストが正解と等しい場合にのみ，加点される．ゆるい採点では，コンパイル時に無視できる無意味なスペース文字は採点対象から除外した（各コードのインデントも無視された）．Table. 6.14 の得点は，コードの順番は採点に考慮していない．しかし，Table. 6.15 は，連続的に正しいかどうかも採点に含む．これは，ある行と次の行が連続で正しい場合，1 点が授与される．Table. 6.16 は，行単位と継続点について学習群間の違いを示す．被験者は各試験で 2 回記憶課題に回

Table. 6.14 Correct line scores of the Recall task. The pr, ps, and wk means Prelearning, Postlearning, and One-week-after exam.

| Code | Mean of strict marking | | | | | | | | | Mean of loose marking | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Reading | | | Visualize | | | Standby | | | Reading | | | Visualize | | | Standby | | |
| | pr | ps | wk | pr | ps | wk | pr | ps | wk | pr | ps | wk | pr | ps | wk | pr | ps | wk |
| #include¡stdio.h¿ | 0.5 | 0.5 | 0.6 | 0.8 | 0.8 | 0.7 | 0.8 | 0.7 | 0.9 | 0.8 | 0.6 | 0.9 | 0.8 | 0.9 | 0.8 | 0.9 | 0.7 | 0.9 |
| int main () { | 0.1 | 0 | 0 | 0.1 | 0 | 0 | 0.2 | 0 | 0 | 0.6 | 0.7 | 0.7 | 0.8 | 1 | 0.7 | 0.8 | 0.8 | 0.9 |
| int a = 0; | 0.3 | 0.3 | 0.2 | 0.3 | 0.3 | 0.1 | 0.5 | 0.4 | 0.2 | 0.9 | 0.9 | 0.9 | 0.9 | 1 | 1 | 0.7 | 1 | 1 |
| int b = 0; | 0.3 | 0.3 | 0.2 | 0.3 | 0.3 | 0.1 | 0.5 | 0.4 | 0.2 | 0.9 | 0.9 | 0.8 | 0.9 | 0.9 | 0.9 | 0.7 | 1 | 1 |
| int i = 0; | 0.3 | 0.2 | 0.1 | 0.3 | 0.2 | 0.1 | 0.5 | 0.4 | 0.2 | 0.8 | 0.8 | 0.7 | 0.8 | 0.8 | 0.9 | 0.7 | 1 | 1 |
| int arr[7] = { 10, 0, -10, 5, 5, 9999, 10}; | 0 | 0 | 0 | 0 | 0 | 0 | 0.1 | 0 | 0.1 | 0.5 | 0.2 | 0.1 | 0.6 | 0.3 | 0.7 | 0.5 | 0.6 | 0.5 |
| for(i=0;i¡=6;i=i+1){ | 0.3 | 0.1 | 0.5 | 0.5 | 0.3 | 0.8 | 0.2 | 0.5 | 0.8 | 0.3 | 0.1 | 0.5 | 0.5 | 0.4 | 0.8 | 0.3 | 0.7 | 0.8 |
| if(arr[i] == 9999){ | 0.2 | 0.2 | 0 | 0.3 | 0.1 | 0.2 | 0.2 | 0.1 | 0.1 | 0.5 | 0.4 | 0.4 | 0.7 | 0.5 | 0.9 | 0.6 | 0.7 | 0.8 |
| break; | 0.8 | 0.6 | 0.6 | 0.8 | 0.8 | 0.9 | 0.8 | 0.8 | 0.8 | 0.8 | 0.6 | 0.6 | 0.8 | 0.8 | 0.9 | 0.8 | 0.8 | 0.8 |
| } | 0.9 | 0.7 | 0.7 | 0.8 | 1 | 0.8 | 1 | 0.9 | 0.8 | 0.9 | 0.7 | 0.7 | 0.8 | 1 | 0.8 | 1 | 0.9 | 0.8 |
| if(arr[i] ¡ 0){ | 0.1 | 0.1 | 0 | 0.2 | 0.3 | 0 | 0.2 | 0 | 0.1 | 0.5 | 0.5 | 0.4 | 0.6 | 0.6 | 0.7 | 0.5 | 0.7 | 0.6 |
| arr[i] = 0; | 0 | 0.2 | 0 | 0.1 | 0.4 | 0.1 | 0.2 | 0.3 | 0.1 | 0.4 | 0.3 | 0.6 | 0.5 | 0.6 | 0.6 | 0.5 | 0.7 | 0.7 |
| } | 0.7 | 0.6 | 0.6 | 0.7 | 0.9 | 0.8 | 0.6 | 0.9 | 0.7 | 0.7 | 0.6 | 0.6 | 0.7 | 0.9 | 0.8 | 0.6 | 0.9 | 0.7 |
| a = a + arr[i]; | 0.1 | 0.1 | 0.2 | 0.2 | 0.3 | 0.1 | 0.3 | 0.3 | 0.2 | 0.5 | 0.4 | 0.4 | 0.6 | 0.5 | 0.5 | 0.6 | 0.7 | 0.5 |
| b = b + 1; | 0.2 | 0.2 | 0.3 | 0.2 | 0.3 | 0.1 | 0.3 | 0.3 | 0.1 | 0.5 | 0.5 | 0.6 | 0.5 | 0.6 | 0.6 | 0.5 | 0.7 | 0.5 |
| } | 0.6 | 0.4 | 0.5 | 0.6 | 0.8 | 0.7 | 0.6 | 0.8 | 0.7 | 0.6 | 0.4 | 0.5 | 0.6 | 0.8 | 0.7 | 0.6 | 0.8 | 0.7 |
| printf(”value is %d\n”, a / b); | 0 | 0 | 0 | 0 | 0.1 | 0 | 0 | 0 | 0 | 0.4 | 0.3 | 0.4 | 0.5 | 0.5 | 0.3 | 0.5 | 0.5 | 0.2 |
| return 0; | 0.5 | 0.4 | 0.3 | 0.5 | 0.7 | 0.4 | 0.6 | 0.7 | 0.5 | 0.5 | 0.4 | 0.3 | 0.5 | 0.7 | 0.4 | 0.6 | 0.7 | 0.5 |
| } | 0.5 | 0.3 | 0.3 | 0.6 | 0.4 | 0.6 | 0.5 | 0.6 | 0.5 | 0.5 | 0.3 | 0.3 | 0.6 | 0.4 | 0.6 | 0.5 | 0.6 | 0.5 |
| (Mean score of all line) | 0.3 | 0.4 | 0.4 | 0.3 | 0.4 | 0.4 | 0.3 | 0.3 | 0.4 | 0.6 | 0.7 | 0.6 | 0.5 | 0.7 | 0.8 | 0.5 | 0.7 | 0.7 |

Table. 6.15 Continuous correct scores of the Recall task

| Learning method | Num of subjects | Mean (SD) of a strict marking | | | Mean (SD) of a loose marking | | |
|---|---|---|---|---|---|---|---|
| | | Prelearning | Postlearning | One-week-after | Prelearning | Postlearning | One-week-after |
| Reading | 11 | 2.55 (2.90) | 4.27 (4.16) | 5.55 (4.77) | 7.73 (4.77) | 10.18 (4.97) | 9.45 (5.58) |
| Visualize | 11 | 2.45 (3.06) | 4.36 (3.67) | 4.18 (3.10) | 6.36 (4.16) | 10.09 (4.48) | 12.09 (4.93) |
| Standby | 10 | 1.70 (1.42) | 3.10 (2.51) | 3.80 (2.32) | 6.90 (3.99) | 10.40 (3.58) | 10.30 (3.69) |

Table. 6.16 Paired *t* test results of correct line and continuous scores in the Recall task. (Bonferroni corrected p-values)

| Comparison | Learning method | Difference (p-value) | | | |
|---|---|---|---|---|---|
| | | correct line | | continuous correct | |
| | | Strict | Loose | Strict | Loose |
| Pre VS Post | Reading | (0.067) | ** (0.004) | (0.067) | * (0.032) |
| | Visualize | *** (<.001) | *** (<.001) | (0.218) | ** (0.002) |
| | Standby | (0.066) | ** (0.001) | (0.273) | * (0.034) |
| Pre VS One week after | Reading | ** (0.005) | (0.524) | * (0.027) | (0.372) |
| | Visualize | *** (<.001) | *** (<.001) | (0.177) | ** (0.007) |
| | Standby | ** (0.003) | *** (<.001) | (0.085) | (0.073) |

答し，この試験では各試験で 2 回目の試験の回答を使用した．5 人の被験者がこの作業で不適切な操作が行われた（コピーアンドペースト）．したがって，彼らの得点は結果より除外した．

Table. 6.17 The strength of the relationship between learning materials and exams (Bonferroni corrected p-values). R means correlation coefficient.

| Learning method | Task | Prelearning exam | | Postlearning exam | |
|---|---|---|---|---|---|
| | | R | P-value | R | P-value |
| Tracing | Reading | 0.69 | * (0.019) | 0.80 | ** (0.002) |
| | Sorting | 0.81 | ** (0.001) | 0.78 | ** (0.003) |
| Visualize | Reading | 0.46 | (0.266) | 0.63 | (0.055) |
| | Sorting | 0.63 | (0.057) | 0.38 | (0.445) |

### (b) 学習セクション

13 人の被験者が読み学習群に属し，12 人が割り当てられた時間内に 61 個の問題を完了した（45 分）．1 人が 14 の問題すべてを完了できなかった（77 %完了）．したがって，被験者の 92 %が活動を完了した．Visualize 群の 13 人の被験者のうち，8 人が割り当てられた学習時間内に 6 つの問題を完了した．3 人の被験者が 1 つの問題を完了できず（86 %完了），そのうち 2 人が 3 つの問題を完了できなかった（57 %完了）．したがって，被験者の 62 %が活動を完了した．完了の評価では，180 秒以上の試行時間を要した回答は完全と見なした．1 人の被験者が非常に短い時間（10 分）内に学習を完了した．

学習教材と試験結果の相関を評価するために, Pearson の相関係数を算出した．結果は Table. 6.17 に示した．いくつかの条件では有意な正の相関を示した．学習活動と試験の得点との関係については，すべての条件下で読み学習群が有意に異なっていた．Visualize 群は差異を示さなかった．

## 6.3.3 考察

結果は学習グループによって異なった．Table. 6.18 は，結果の要約を示す．トレース課題の結果は問題 1〜4 と 8 からのものであり，並び替え課題からのものは問題 1〜3 からのものであり，これらは学習教材の最後の問題に対する関連性が低い．記憶課題の結果は，正しい行の得点と継続点の合計であり，各得点は厳密な採点とゆるい採点の平均とした．（例えば，厳密採点とゆるい採点の結果はそれぞれ***と*である場合は，平均は [**] で示す．もしも結果が，それぞれ**と*の場合は，平均は [*.] で示す．）

統制群の被験者は，他のグループが学習活動に従事している間は制限を課さずにいたため自由にしていた．2 人の被験者が約 5〜10 分間プログラミングに関連する課題を学習を行ってしまった．ただしその学習時間と被験者数は，他のグループの半分以下であり，影響は限定的で

Table. 6.18 Summary of *t* test result. The pr, ps, and wk means Prelearning, Postlearning, and One-week-after exam.

| Learning method | Comparison | *t* test result for each task | | |
|---|---|---|---|---|
| | | Tracing | Sorting | Recall |
| Reading | pr vs ps | * | | *. |
| | pr vs wk | * | * | *. |
| Visualize | pr vs ps | | * | **** |
| | pr vs wk | | | **** |
| Standby | pr vs ps | | ** | *. |
| | pr vs wk | | *** | **. |

ある．統制群については，一部の課題にて他の学習群と同等か同じ程度の得点像が見られた．この解釈については，次のセクションで検討する．

### (a) 各課題の結果

読み学習群は，トレース課題において良い結果を示し，この結果は 1 週間後も変わらなかった．したがって，取得された知識は少なくとも 1 週間維持された．トレース課題の最も困難な問題（例えば，5–7）の結果は 1 週間後に変化したが，トレーススキルを効果的に改善したとみなせる．ただし，ソートアルゴリズムのような難しいコードでは知識が忘れられている可能性が見られる．Visualize 群は読み学習群よりも優れた結果とはならないが，一定の範囲で有効性は見られた．統制群は，学習後試験では変化が見られなかったが，1 週間後の試験で得点が改善された．この結果は，本実験以外の授業や自己学習により，ある程度影響を受けた可能性が考えられる．

コード並び替えスキルは，読み学習群については得点の向上が限定的であるため，効果が見られない．Visualize 群はこのスキルがある程度向上したが，得点は 1 週間で減少した．対照的に，読み学習群は，1 週間後の試験で改善を示した．これは，本実験以外の授業や自己学習の効果とも考えられるが，Visualize 群は同様の傾向が見られない以上は，読み学習の活動から得られた知識が，授業や自己学習の効果を変えた可能性を示している．統制群はこの最も良い結果を示した．しかし，事前試験と学習後の問題が似ており，選択課題に表示された 13 のコードがヒントとなったと考えることができる．特に統制群は，他の群が学習中にはプログラムコードが提供されまなかった．この追加情報がないということが事前試験の記憶を明確したため，並び替え得点が向上したと解釈することが可能である．

記憶課題の結果は解釈が容易であり，読み学習群と統制群の効果には一定の範囲であるが，Visualize 群はもっともよい結果を示した．これは，コードを書く活動が Visualize 群に含まれ

ており，それが効果的にコード記憶を向上させ，高い得点につながったと考えられる．これは，コンパイルを通して反復で試行錯誤を行った結果（すなわち，文法エラーのメッセージを読み取り，コードのエラーを修正し再度実行）と考えられる．

### (b) 学習方法や活動の特徴

読み学習群はトレーススキルの改善を示し，並び替えスキルは学習後 1 週間で改善された．Visualize 群は，コード記憶のスキルの向上を示した．しかし，コード並び替えスキルはある程度改善されたが，1 週間以内に忘れられた．統制群の結果は，学習なしで一部の問題の得点が向上したことを示した．この傾向は，コード並び替え課題では明らかであるが，コードトレース課題では見られなかった．あえて学習を行わない（追加情報が提示されない）ことによる結果であると見なすことができる．これらの結果は，学習活動が学習教材の種類によって異なり，異なる活動がプログラミング時に払われる注意に影響し，その注意の違いはスキルの獲得に影響すると考えられる．言い換えれば，コードトレースやコード並び替えなどのプログラミングの多様なスキルは，単一のタイプの学習教材を使用することによって最適に改善されないことを示しており，さらに，学習後の知識の向上は，注意の方向に影響を与えると考えられる．

読み学習群は，学習完了率が他の学習方法よりも良いため，時間制御が容易という特徴がある（例：すべての学習者が 30 分以内に終了できる課題を含む教材が作りやすい）．対照的に，Visualize 群のメンバー間では学習活動時間に差があり，1 人の被験者は脱落したと考えられる．読み学習のような時間管理が容易であるという特徴は，授業で使用しやすいだけではなく，宿題や自習に使用される教材としても優れている．このような特徴から示される点は，学習者が自身で学習を進めることが容易であることを意味する．また，読み学習群の学習得点と試験の得点の強い相関関係は，この学習手法の拡張性において期待が持てる結果と考えられる．例としては，各学習者にカスタマイズされた問題を提供するような，アダプティブラーニングシステムと併用することも可能であると考えられる [6].

実験のまとめとしては，併用する学習方法が有効であることを示している．プログラミングクラスの初期学習としては読み学習を行い，選択，反復，およびそれらの組み合わせなどの基本的な処理を学ぶことが好ましいと思われる．トレーススキルの改善後，Visualize や書き課題などの従来の方法を使用して，学習者が書きスキルを取得できるようにする．このような方法を組み合わせることで，学習者はトレースや書きスキルの両方を向上させることができ，ドロップアウト率を下げることができると考えられる．

### 6.3.4 まとめ

読み学習，Visualize 学習，統制群で比較実験を行った．その結果，読み群ではコードトレーススキルが向上し，Visualize 群ではコード記憶スキルが向上した．したがって，学習者のプログラミングへの注意のむき方が，適用される学習方法のタイプによって影響され，結果として改善するスキルに影響を与えると考えられる．したがって，異なるスキルを同等に向上させるためには，さまざまな種類のプログラミング活動が必要である．この結果は，トレーススキルはプログラミングにおける中核のスキルであることを踏まえると，読み学習が初心者への適切な導入教材であることも示している．将来の課題としては，長期的な実験における読み学習と Visualize 学習の組み合わせについて効果を検討する必要がある．

## 参考文献

[1] Katherine B. McKeithen, Judith S. Reitman, Henry H. Rueter, and Stephen C. Hirtle. Knowledge organization and skill differences in computer programmers. Cognitive Psychology, 13(3):307 – 325, 1981.

[2] Simon. Soloway's rainfall problem has become harder. In 2013 Learning and Teaching in Computing and Engineering, pages 130–135, March 2013.

[3] Herbert Schildt. Teach Yourself C. McGraw-Hill Osborne Media, 1997.

[4] Philip J. Guo. Online python tutor: Embeddable web-based program visualization for cs education. In Proceeding of the 44th ACM Technical Symposium on Computer Science Education, SIGCSE '13, pages 579–584, New York, NY, USA, 2013. ACM.

[5] C tutor. http://www.pythontutor.com/c.html.

[6] Dusan Jovanovic and Slobodan Jovanovic. An adaptive e-learning system for java programming course, based on dokeos le. Computer Applications in Engineering Education, 23(3):337–343, 2015.

## 6.4 実践的調査の知見総括

実践的調査の知見を総括すると，学習形式については出力結果選択形式で，かつ，問題に取り掛かれる程度以上の説明を行わない手法が好ましいと考えられる．理由として，実験４において同一条件下であれば，書きより読み学習の効果が高い傾向が示されたことと，実験５において出力結果選択形式で，かつ，説明の省略手法が，理解型学習（説明 +Visualize+ 書き）よりプログラミングにおいて中核的なトレーススキルの向上を示したためである．

出題形式については，典型的なコードの反復的な出題が好ましいと考えられる．理由として，実験４において満遍なく読みを行う学習の記憶が時間とともに曖昧化する傾向が見られたことと，実験５において典型的なコードの反復的な出題に効果が見られたためである．この典型的なコードの出題については，学習する要素が多い場合に広く浅くなってしまう出題内容の問題を解決し，制限された時間内で反復数を増加する点に効果的な手法であった．

以上のような経験型教材が好ましいが，しかしながら別の手法の学習も組み合わせた方がよいという結果も見られた．実験５においてコードの記憶課題は，Visualize を含む理解型学習の結果が最も優れていたことから，一つの手法が万能というわけではないことが示されている．書く活動に関しては記憶を明確にする可能性が示されたが同時に学習者により多様性が見られてしまい効果の統一が容易ではない結果となったため適用方法に注意が必要である．

動機づけという観点では，選択形式の連続出題において誤答で正答表示を行い，誤回答の明確化をすることが効果的であった．理由として，実験５において比較対象であった理解型学習（説明 +Visualize+ 書き）で，低得点にも関わらず極端に学習時間が短い，脱落とみなせる学習者がおり，選択形式の連続出題を行う経験型学習ではそのような結果とならなかったためである．

その他には，理論的な調査の結果を支持する結果が現れた．文法知識は短時間の学習では時間とともに忘れられる傾向が見られたため，実験２と同様の結果となった．反対に，書きの知識は時間とともに減少しなかったため，個別の知識を忘却しても影響を受けないと言う意味で実験３と同様に演繹以外の推論が用いられた可能性が示された．

# 第 7 章

# 調査結果から導かれる経験に着目した学習方法の提案と効果検証

## 7.1　調査結果から導かれる学習方法

調査により得られた知見を踏まえ，次の学習方法を提案する．

1. 文字ベースの教材（ブロックベースではない）
2. 学習要素の説明は問題に取り掛かれる範囲まで（それ以上は行わない）
3. 読み形式の連続出題 (出力結果選択形式)
4. 学習問題の正誤即時判定
5. 誤回答の際に正答を表示
6. 単純な操作で学習可能
7. 低難度の問題の出題 (三択問題，誤回答の明確化)
8. 出題内容はスキーマを意識した形態 (反復的な出題，類似問題)
9. 出題の観点は「出力 → 他連接処理 → 反復, 分岐条件」
10. 理解型教材と経験型教材（上記 1–9 の特徴の教材）の併用

学習形態には，実験４の結果を参考に読み学習とし，実験５の結果が良好であることから，文字ベースの連続出題でできるだけ多数の回答を行える形式，説明を初めに行わない方式とする．また，実験２にてスキーマ形式の特徴が確認されたことから，応用力の向上を目的として，デフォルト値とスロットの獲得を狙う出題形式とする．これは，デフォルト値となることを期待する代表的なコードと出力結果の問題（中心問題）を設定して，それを反復的に出題する形式と，変数化を期待して中心問題の一部分を変更した問題を出題する形式の組み合わせと

なる．簡単にその形態を反復処理の例として示すと次の形になる．

1. 中心問題 (配列の順出力)
2. 類似問題 1(出力内容の変更)
3. 中心問題
4. 類似問題 2(演算処理の追加)
5. 中心問題
6. 類似問題 3(反復条件の変更)

となる．一問おきに中心問題が繰り返し出題される方式である．また，中心問題以外も全く異なる種類ではなく，変数化を期待した一部分のみを変更した類似問題となる．これにより，デフォルト値とスロットの獲得を促す．そして，その他の特徴としては，実験５とは異なり例題表示を取りやめた．理由として，できるだけ多くの問題を回答するにあたり，例題を提示しない方が単位時間の問題挑戦量が多くなると見込めることと，サンプルコードを参考にそれを改変すると言う類推的な回答については，デフォルト値となる中心問題がその代わりとなることが期待できるためである．そして，実験５において読みの反復手法であってもすべての能力を向上させることができるわけではなく，学習手法により向上する能力が異なる可能性が示された. したがって経験に着目した学習方法はそれ単体で十分ではなく，また並列性の能力使用の点から考えても，従来的な理解を重視する学習も組み合わせることが必要である．したがって，上記の学習手法を終えた後に，従来方式の説明を行うこととする．

中心問題に使用されるコードの例と一部を変更したコードの具体例は次の通りとなる．

```
// 中心問題
#include<stdio.h>
int main(){
    int i, arr[5]={4,3,2,6,8};
    for(i=0;i<5;i++){
        printf("%d," , arr[i]);
    }
    return 0;
}
```

```
// 類似問題1 （出力内容の変更）
#include<stdio.h>
int main(){
    int i, arr[5]={4,3,2,6,8};
    for(i=0;i<5;i++){
        printf("arr[%d] = %d," , i, arr[i]);
    }
```

```
    return 0;
}
```

```
// 類似問題2 （演算処理の追加）
#include<stdio.h>
int main(){
    int i, arr[5]={4,3,2,6,8};
    for(i=0;i<5;i++){
        arr[i] = arr[i] + 1;
        printf("%d," , arr[i]);
    }
    return 0;
}
```

```
// 類似問題3 （反復条件の変更）
#include<stdio.h>
int main(){
    int i, arr[5]={4,3,2,6,8};
    for(i=1;i<3;i++){
        printf("%d," , arr[i]);
    }
    return 0;
}
```

授業などを通して学習を行う場合，反復学習については，授業内で行うことが理想ではあるが，自習や宿題などの形式として授業前に済ませておくと言う方針も可能な学習方法である．

## 7.2 提案手法の学習効果検証（実験６）

### 7.2.1 提案する学習環境の制作

#### (a) 設計

この学習環境の重要な要素は，反復的な多項選択問題を持つプログラミングスキーマを作成することである．また，誰にとっても便利なシステムを開発するためには，使いやすく，さまざまな条件に適応することも重要である．

**■反復的な選択問題を容易に作成するための機能** スロットとデフォルト値で構成されるプログラミングスキーマを作成するには，学習トピックのメインプログラミングコードを設定して，それに関する問題をに繰り返し出題する必要があることが考えられる．その後，スロットとなるメインコードとは少し異なる問題が，メインコードの問題の間に出題される．このような問題を容易にに作成するには，自動問題順設定機能が必要である．まず，ユーザーはいくつかの問題を作成し，そしてトピックのメインコードの問題を選択する．次に，自動出題順選択機能を使用して出題順を設定する．この出力された順番は最後に変更することが可能である．

**■使いやすさ** システムによっては，さまざまな種類の問題を作成する機能を持たせることは可能であるが，その場合の操作は複雑で使いづらくなる．このシステムは問題を容易に作成するために設計されており，選択肢形式の問題のみの作成が対象となる．作成手順は，1）問題と正解の登録，2）ダミー回答の登録，3）ステップ 1 と 2 の繰り返しによる問題の登録，4）メインコードの問題を選択，5）問題の順序の選択の 5 つのステップである．学習者は問題に答えるためのアカウントを作成できるが，ログインせずに匿名で回答することも可能である．

**■時間，場所を選ばずに学習** 学習環境は，オペレーティングシステムまたはインストールを必要としないウェブベースのシステムが好ましい．また，スマートフォンのアクセスを許可することで，学習者は時間と場所を選ばずに問題に答えることができる．試行時間が短い場合，学習者は簡単に課題を繰り返すことができる．このような活動は，マイクロラーニングなどの他の学習方法と互換性がある [1].

**■プログラミング学習システムとしての機能** プログラミングコードを自動的に色分けして問題などに表示することで，学習がしやすくなる．

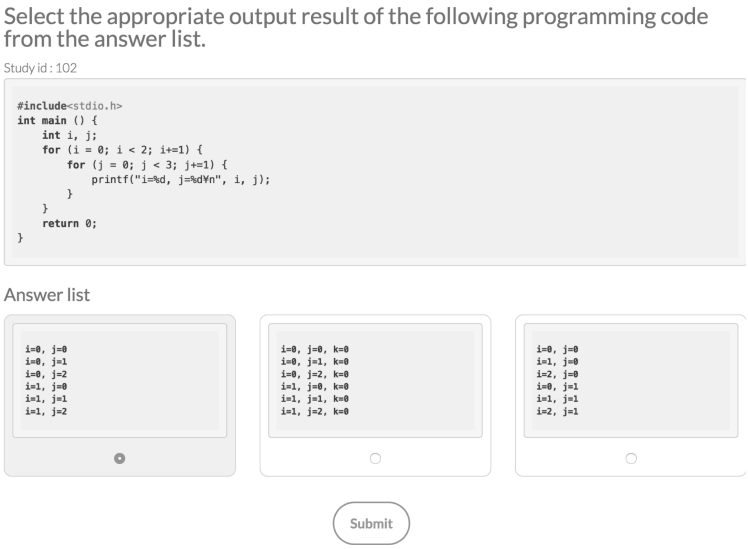


Fig. 7.1 Screen of a multiple-choice question.

**■授業での使用** このシステムは公開されたウェブサイトであり，ユーザーは自由に問題を作成して回答できる．しかし，そのようなウェブサイトは，授業で使用する場合，学生だけの結果を集計することが難しいため，宿題や授業での使用が難しい．したがって，システムはプライベートな問題を作成する機能を有する．ユーザーが問題作成中にプライベート問題と設定すると，アクセス URL が生成される．そして，URL を知っているユーザーだけが問題にアクセスでき，ユーザーが識別やコメントを入力できるテキスト領域が問題の先頭に表示される．

### (b) 実装

サーバーオペレーティングシステムは Amazon Linux 上に作成し，システムは AWS サービス上で実行される．Web サーバーに Nginx を使用し，サーバー側システムは PHP によって開発した．PHP フレームワークの Laravel が開発に使用され，データベース管理システムに MySQL を使用した．Bootstrap CSS フレームワークがフロントサイトデザインに適用し，ほとんどの画面でスマートフォンアクセスを可能にするレスポンシブデザインを採用した．すべての通信は https 通信で暗号化され，コンテンツ配信ネットワークサービスを配信に使用し，サイトにすばやくアクセスできるようにした．Fig. 7.1 は，完成したシステムの複数選択問題の画面を示す．

### 7.2.2 実験内容の詳細

実験[*1]では，提案された学習システムを評価するために比較実験を採用する．比較対象は次の通りとなる．

1. 出力結果選択形式の連続出題による読み中心の経験型学習（反復出題学習）
   （a）文字ベースの教材（ブロックベースではない）
   （b）学習要素の説明は問題に取り掛かれる範囲まで（それ以上は行わない）
   （c）読み形式の連続出題 (出力結果選択形式)
   （d）学習問題の正誤即時判定
   （e）誤回答の際に正答を表示
   （f）単純な操作で学習可能
   （g）低難度の問題の出題 (三択問題，誤回答の明確化)
   （h）出題内容はスキーマを意識した形態 (反復的な出題，類似問題)
   （i）出題の観点は「出力 → 他連接処理 → 反復, 分岐条件」
   （j）理解型教材と経験型教材（上記 a–i の特徴の教材）の併用
2. 説明と Visualize ツールと書き学習を組み合わせた理解型学習（Visualize 学習）
   - 教科書通りの説明
   - Visualize で処理確認（デバッガ形式のツール）
   - 書き課題

実験の手順を Fig. 7.2 に示す．

最初に，被験者は事前試験を受け，その後，合計試験得点が出来るだけ等しくなるように 2 つの学習群に分類した．一つは反復問題学習システムを用いて学習し，他は Visualize 学習システムによって学習した．すべての被験者は，その後，事後テストを受けた．被験者は学習と試験の間に 5 分の休憩を取った．

この実験は，Web ベースのシステムで実行された．反復問題システムを除いて，システム全体がこの実験のためにのみ開発された．したがって，被験者は実験中にウェブブラウザのみを使用した．

[*1] ©2018 IEEE. Reprinted, with permission, from Satoru Kikuchi and Kazuhiko Hamamoto, Implementation and Pilot Evaluation of an Iterative Learning Environment for Programming Education, 2018 International Conference on Learning and Teaching in Computing and Engineering (LaTICE), 06/2018.

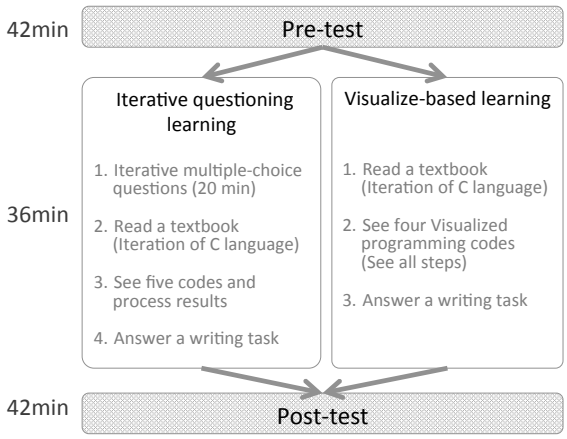


Fig. 7.2 A procedure of the experiment.

学習対象は，C 言語のネストされた反復コードとした．コードは深くネストされた反復（すなわち，二重，三重，および四重のネスト）からなる．初心者プログラマにとり，容易ではない内容 [2] であり，かつ馴染みがない内容となる．

### (a) 事前試験と事後試験

事前試験と事後試験は 3 種類の問題で構成されている．問題の詳細と時間制限は，Table. 7.1 と付録に示されている．最初の 2 つの問題（Write1 と Write2）は，サンプルコードなしの書き課題である．被験者は問題要件を満たすためにプログラミングコードをテキスト領域に書き込む必要がある．被験者は，コンパイルボタンを使用して入力コードをコンパイルし，コンパイルエラーメッセージがあればそれを見ることができるが，出力結果は表示されない．次の問題（Write3 と Write4）は，最初の 2 つの問題に類似した同じコンパイルボタンを使用した課題であるが，単純な反復サンプルコードが対象に表示され，文法のヒントが提供された．筆記課題の後，被験者は 4 つまたは 5 つの問題に回答した（Trace5 から Trace9）．被験者はプログラミングコードを見て，処理結果をテキスト領域に書き込む．各課題には時間制限が設定されていて，時間切れとなった場合は，被験者は答えを入力できない仕様とした．

三重のネストされたコードが初心者プログラマにとっては難しいと仮定されているため [2]，問題（Write1 から Write4，Trace5 から Trace7 まで）は，三重か四重の反復と選択処理を含むコードとした．事前と事後の両試験で同様のコードを使用した．このような問題は，多くのワーキングメモリを必要とし，プログラミングスキーマを分析するのに適していると考える．事後試験の Trace8 と Trace9 には，適用されたスキルを分析するための選択と反復の組み合わせコードが出題された．事前テストの Trace8 は，選択の理解レベルをチェックするための分

Table. 7.1 Questions type, time limit and description of the pre-test and post-test.

| Pre test ID | Post test ID | Task | Limit (sec) | Description |
|---|---|---|---|---|
| 1 | 1 | Write[a] | 1080 | Triple nested iteration.[c] |
| 2 | 2 | | | Triple nested iteration & caluculation. |
| 3 | 3 | Write[b] | 840 | Quadruple nested iteration. |
| 4 | 4 | | | Triple nested iteration & caluculation |
| 5 | 5 | Trace | 600 | Triple nested iteration. |
| 6 | 6 | | | Triple nested iteration & caluculation |
| 7 | 7 | | | Triple nested iteration. |
| 8 | - | | | Selection statement. |
| - | 8 | | | Triple nested iteration & selection. |
| - | 9 | | | Triple nested iteration & selection. |

[a]Code writing task without a sample code.
[b]Code writing task with a sample code.
[c]Quadruple at the post-test.

Table. 7.2 Question type and order of Iterative questions group.

| ID | Type | ID | Type | ID | Type | ID | Type | ID | Type |
|---|---|---|---|---|---|---|---|---|---|
| 1 | a | 6 | a | 11 | g | 16 | B | 21 | A |
| 2 | b | 7 | e | 12 | a | 17 | A | 22 | E |
| 3 | c | 8 | a | 13 | h | 18 | C | 23 | A |
| 4 | a | 9 | f | 14 | a | 19 | A | | |
| 5 | d | 10 | a | 15 | A | 20 | D | | |

[a]Main code 1 : A code of iteration and output.
[b−h]Arranged code of a.
[A]Main code 2 : A code of iteration and calculate.
[B−E]Arranged code of A.
[a,e,f,A,C] The codes used as a Visualize code.

岐処理が出題された.

### (b) 学習

**■反復出題学習** 反復出題学習は 2 つの部分で構成される．最初のパートは反復問題に答える 20 分の学習である．このパートでは，提案された学習システムが適用され，問題は，出力と演算が対象でネストされた反復処理を学習するように設計されている．問題は 23 問あり，被験者が 23 問目の回答をした後は再度問題 1 が出題される．この部分は 20 分間続き，プログラミングスキーマの作成を狙いとして同じ問題が繰り返し出題される．問題の順序と詳細コードは Table. 7.2 と付録に示す.

次の学習パートは理解学習と書き課題で，16 分の時間が当てられた．被験者はまず，C 言

語の教科書 [3] の日本語版の記載で，C 言語の反復処理の記述を読む．その後，5 つのコードと出力結果が示され，前の反復パートのコードを確認する．その後，書き課題を行う．被験者は，問題の要件を満たすコードをテキスト領域に書き込む．被験者はコンパイルボタンを使用して，コンパイルエラーと入力コードの出力結果を確認することができる．

**■Visualize 学習** Visualize 学習群は，学習および筆記課題のために 36 分が当てられた．被験者は最初に反復処理の説明（反復出題学習群と同一）を見た後，変数の値と各行の出力を示す Visualize プログラミングコードで学習する．これは公開されたシステム [4] から作成された．その後，被験者は反復出題学習群と同じ書き課題に回答した．Visualize に使用されるコードは，Table. 7.2 に示す反復出題学習のコードから選択され，詳細は付録に示す．

### 7.2.3 結果

#### (a) 全体的な結果

本実験について，被験者の募集にあたり 2 時間程度の拘束となるため 2,000 円の金券を報酬として用意した．また，その他に報酬として実験後に希望者にはスマートフォンアプリケーションと Web システムの作成について実技を交えて説明する実践的な講義を行った．実験には 22 名の被験者が参加し，全員が大学のプログラミングコースを受講した経験がある．9 人は大学新入生，7 人は 3 年生，3 人は 4 年生，3 人は大学院生であった．実験はすべての被験者が一度に出席する事が難しいため，計 3 回実行された．すべての被験者は，この実験に参加するための図書カード 1,500 円分を実験後に謝礼として受け取った．

各問題ごとに 1 点を割り当てたため，事前試験の最大得点は 8 であり，事後試験は 9 である．問題の配点の詳細を Table. 7.3 に示す．

事前試験における書き課題の平均点は 0.90 (SD=0.98) であり，トレース課題は 1.97 (SD=0.82) であった．事後試験の得点は 2.04 (SD=1.09) と 2.21 (SD=0.84) であった．

#### (b) 学習群間の比較

事前試験の後，被験者は学習群間の得点の偏りに配慮し 2 つの学習群に分類され，11 人の被験者が各学習群に振り分けられた．Table. 7.4 に事前テストの得点を示す．我々は比較のために Welch の $t$ 検定で両側検定を採用した．学習群間でいずれの課題でも有意差は見られなかった．

学習の効果を調べるために，事前試験と事後試験との比較を分析した．問題 1–7 は試験前

Table. 7.3 Point allocation details for each question. In writing task, the output results of a written code is scored. In tracing task, the written output code is scored.

| Score | Description |
|---|---|
| 0.333 | The score of formats. Subject get the full point if the format of an output result (e.g. line break, space, and comma) is correct. If the format is not correct, the subject gets no point. ( Only the format is a target of the scoring. The output value is not scored. ) |
| 0.333 | Score of output number. (e.g. In case correct answer has five outputs, subject get full points when the answer has five outputs and get 80% points when the answer has 4 or 6 outputs.). The unit of counts is based on output functions usage in the writing task and line breaks in the tracing task. |
| 0.333 | The score of output values. (e.g. In case correct answer has five outputs, subject get full points when the answer has five correct values and get 80% points when first four values are correct. If there is a wrong value in the answer, following values in the answer is not scored. ) |

Table. 7.4 Scores of pre-test between the learning groups.

| Task | Mean (Standard Deviation) | | t | df | $p$ |
|---|---|---|---|---|---|
| | Iteration | Visualize | | | |
| Write | 0.57 (0.85) | 1.22 (1.09) | -1.568 | 18.9 | 0.13 |
| Trace | 2.08 (0.77) | 1.85 (0.93) | 0.657 | 19.3 | 0.52 |
| All | 2.66 (1.34) | 3.07 (1.96) | -0.577 | 17.7 | 0.57 |

と試験後の間で類似しているので，得点を比較することで学習の効果を調べることが可能である．

反復出題学習群の結果を Table. 7.5 に示す．これは，両側検定の対応のある t 検定を用いて示された，各問題の事前試験と事後試験との得点差を示す．多重検定の対応として Bonferroni 補正が使用された．Write1 と Write4 は，$p = 0.05$ のレベルで有意差を示した．書き課題とすべての問題の合計は，$p = 0.01$ のレベルで有意差を示した．Table. 7.6 は，Visualize 学習群の結果であり，Table. 7.5 と同じ分析を示す．Write2 では，書き課題とすべての問題の合計が $p = 0.05$ のレベルで有意差を示した．書き課題の得点は両方の学習によって改善され，トレース課題は改善されなかった．書き課題の違いの詳細を調べるために，Fig. 7.3 に示すボックスプ

Table. 7.5 Comparison of points between pre-test and post-test in Iterative learning group. (Bonferroni corrected p-values)

| Task | Mean (Standard Deviation) | | t | df | $p$ |
|---|---|---|---|---|---|
| | pre-test | post-test | | | |
| Write1 | 0.17 (0.34) | 0.59 (0.32) | -3.210 | 10 | 0.019* |
| Write2 | 0.12 (0.23) | 0.42 (0.35) | -2.460 | 10 | 0.067 |
| Write3 | 0.18 (0.32) | 0.43 (0.39) | -2.097 | 10 | 0.125 |
| Write4 | 0.10 (0.25) | 0.55 (0.34) | -2.906 | 10 | 0.031* |
| Trace5 | 0.84 (0.20) | 0.84 (0.24) | 0.085 | 10 | 1.000 |
| Trace6 | 0.73 (0.29) | 0.73 (0.39) | 0.009 | 10 | 1.000 |
| Trace7 | 0.51 (0.40) | 0.59 (0.38) | -0.769 | 10 | 0.920 |
| Write All | 0.57 (0.85) | 1.99 (1.07) | -3.688 | 10 | 0.008** |
| Trace All | 2.08 (0.77) | 2.16 (0.93) | -0.403 | 10 | 1.000 |
| All | 2.66 (1.34) | 4.15 (1.42) | -4.075 | 10 | 0.004** |

*Results are significantly different at $p < 0.05$.
**Results are significantly different at $p < 0.01$.

Table. 7.6 Comparison of points between pre-test and post-test in Visualize learning group. (Bonferroni corrected p-values)

| Task | Mean (Standard Deviation) | | t | df | $p$ |
|---|---|---|---|---|---|
| | pre-test | post-test | | | |
| Write1 | 0.29 (0.38) | 0.61 (0.33) | -2.557 | 10 | 0.057 |
| Write2 | 0.15 (0.27) | 0.50 (0.33) | -3.531 | 10 | 0.011* |
| Write3 | 0.38 (0.33) | 0.51 (0.34) | -1.719 | 10 | 0.233 |
| Write4 | 0.40 (0.32) | 0.47 (0.31) | -1.119 | 10 | 0.578 |
| Trace5 | 0.74 (0.29) | 0.78 (0.23) | -0.736 | 10 | 0.957 |
| Trace6 | 0.64 (0.41) | 0.79 (0.40) | -1.616 | 10 | 0.274 |
| Trace7 | 0.47 (0.35) | 0.69 (0.36) | -1.728 | 10 | 0.230 |
| Write All | 1.22 (1.09) | 2.09 (1.21) | -3.420 | 10 | 0.013* |
| Trace All | 1.85 (0.93) | 2.26 (0.82) | -1.963 | 10 | 0.156 |
| All | 3.07 (1.96) | 4.35 (1.95) | -3.095 | 10 | 0.023* |

*Results are significantly different at $p < 0.05$.

ロットと散布図による結果も分析した．

事後試験の Trace8 と Trace9 は，ネストされた反復処理と分岐処理を使用する事により，応用力を調査したものとなる．また，事前試験の Trace8 は分岐処理の理解を図るための確認問題となる．結果は Fig. 7.4 に示す．

反復出題学習の 6 人の被験者は，簡単な選択問題（事前テストの Trace8）に正答し，9 人の被験者が Visualize 学習群で正答した．学習後，正解の数は両方の学習群のすべてのトレース課題で減少した．特に，事後試験の Trace8 および 9 では，得点は Trace5–7 の得点よりも低

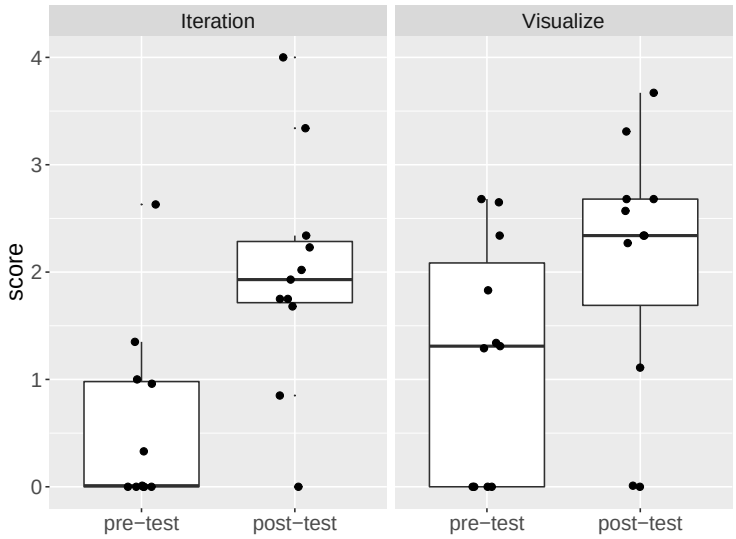


Fig. 7.3 Boxplot and scatter plot of the writing task for both interventions.

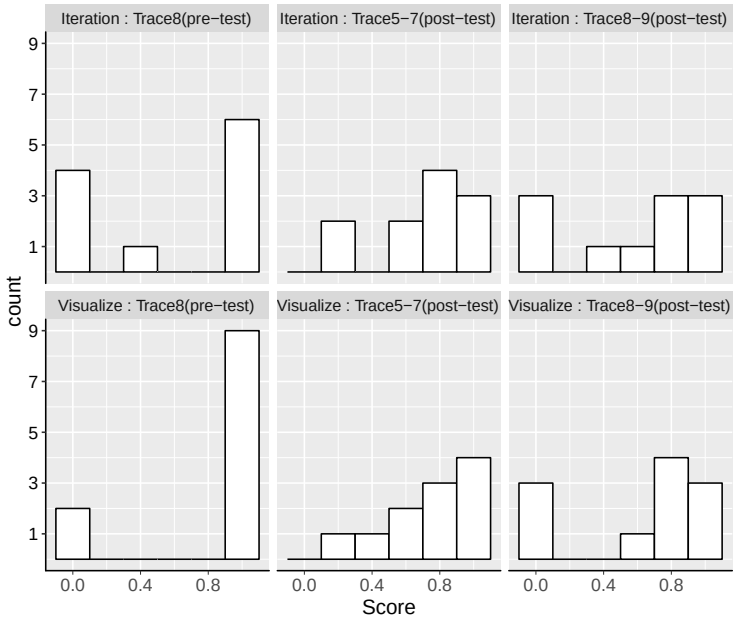


Fig. 7.4 Histogram of Trace8 in pre-test, Trace5–7 in post-test and Trace8–9 in post-test for both Interventions.

かった．Trace5–7 の中間的な得点を受けた被験者は，Trace8–9 の得点が低かった．この傾向は両群で同一であった．

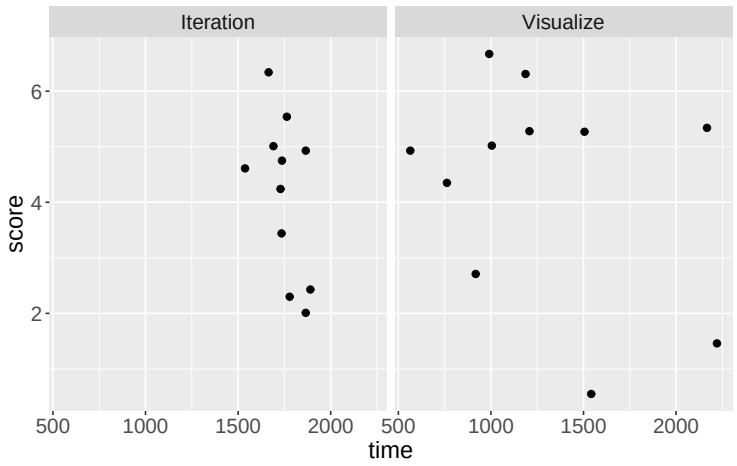


Fig. 7.5 Scatter plot of the learning time and post-test score of each interventions.

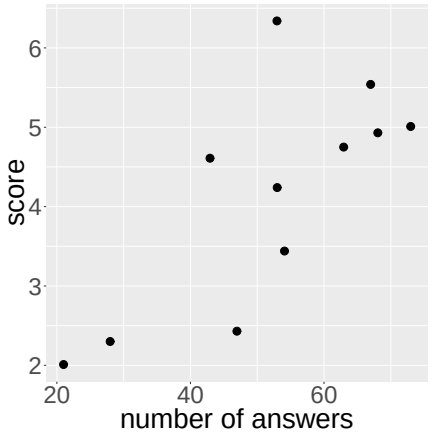


Fig. 7.6 Scatter plot of the answered number at intervention of iterative learning and the post-test score.

### (c) 手法の違いによる学習時間と学習進度

被験者が学習の制限時間内に最終問題に答えたら学習セクションは終了し，セクション時間が終了するのを待つ必要があった．そのため，各被験者は学習を同時に開始したが，終了時間は被験者によって異なる．学習の特徴を調査するために，学習時間および試験後得点を分析した．Fig. 7.5 は，各学習の学習時間と試験後の得点の散布図を示す．反復出題学習群の Pearson の相関係数は，-0.53（$p = 0.09$）であり，Visualize 学習群は-0.34（$p = 0.31$）であった．

反復出題学習の問題は，被験者は 20 分の時間制限内でできるだけ多くの問題に回答した．したがって，回答される問題の数は被験者によって異なる．Fig. 7.6 は，回答数の散布図と各学習の試験後の得点を示している．相関係数は 0.75（$p$ <0.01）であった．

### 7.2.4 考察

どちらの学習群も様々な能力レベルの被験者を含むため，高低の得点に広がる結果となった．事前試験の得点は，2 つの学習群の間でほぼ同じレベルであった．書き課題の平均得点は，お互いにわずかに異なっていた（すなわち，0.57 対 1.22）が，有意差がなかったため結果に影響しない．

どちらの学習方法も書き課題の得点に正の影響を与え，事後試験の点数は同じレベルに改善した．反復出題学習群は，書き課題の試験前と試験後との間の t 検定の信頼性がより良い結果を示した．どちらの学習群でもトレース課題は改善されなかった．各問題を分析すると，反復出題学習は 2 つの問題（Write1，Write4）で，Visualize 学習群は 1 つの問題（Write2）で改善された．したがって，結果は，反復出題学習が本実験の Visualize 学習と同じまたはより有効であることを示している．さらに，反復出題学習は，Fig. 7.5 に示すように，事後試験において低得点の被験者が少ないため，最低得点群の得点を底上げすることが期待される．

応用力を調査するために設計された選択と反復の問題の組み合わせの結果は，学習群間で差は見られなかった．Fig. 7.4 で示される通り，選択処理のトレース課題（事前テストの Trace8）で正答した被験者の数が Visualize 学習群で多かったが，反復処理のトレース課題の正答人数と，複合処理のトレース課題の正答人数がほぼ同じであった．どちらの学習群においても，被験者が反復処理のトレース課題および書き課題で高い得点を得られなかった場合，それらは複合処理のトレース課題に対して正解を得ることができなかった．

学習時間と得点の分析では，Visualize 学習群では十分な時間を取らずに学習する被験者がいる傾向が示された．これは，被験者によっては学習時間の分散が大きく，学習時間と学習後試験との相関係数が低いためである．反復出題学習群は制限時間内で反復出題学習を行った．学習時間は被験者間では同じであったが，制限時間までは自身のペースで回答した．これは，低得点の学習者において一定の学習量を担保できるため，理解の向上が促進できると考えられる．さらに，反復出題学習群の相関係数は，学習時の回答数と学習後試験の得点との間で高かった．これは，学習中に学習者の学習進捗状況により出題を調整する事が行いやすい特徴があり，反復出題学習方法の拡張性を示している．拡張例としては，適応型学習システムなどの別の方法を組み合わせて学習者のレベルに合ったカスタマイズされた問題を提供することが考えられる [5].

### 7.2.5 まとめ

本研究では，プログラミングスキーマの作成に焦点を当てた学習環境を提案した．このシステムを実装し，それを他の学習環境，すなわち Visualize 学習環境と比較することにより評価した．提案されたシステムは，反復的な選択肢問題を容易に作成できるように設計されており，無料のオンライン学習ツールとして公開された [6].

その結果，以下のことが示された．1）提案手法は，書き課題の得点が同じレベルに向上したため，Visualize 学習環境と少なくとも同程度の効果があると考えられる．また，提案手法は，$t$ 検定の信頼度が高いため，より効果的である可能性がある．2）いずれの学習環境においても，トレース課題の得点は改善されなかった．3）提案手法は Visualize 学習環境よりも低得点群を改善する傾向があることが示された．4）学習で学んでいない内容に関する問題の得点はいずれの群も改善されなかったため，プログラミングの応用スキルに改善は今回は見られなかった.

トレース課題の結果が改善されなかったため，読み能力の向上については，実験５で確認がされておりこれとは異なる結果となった. 学習時間が短い点もあるため，短時間で学習効果が見込める学習内容とそうでない学習内容がある可能性が示された.

また，応用スキルについては改善が見られなかったが，こちらも学習時間に起因する可能性があるため，今後の課題として，このシステムを使用してプログラミングクラスの初級レベルに適用できるよう，学習教材の量を充実される必要がある．そして，より多い人数で長期的な実験を用いて提案する学習システムの特徴について応用スキルを中心に調査が必要となる．

## 参考文献

[1] Harrison Hao Yang. New world, new learning: Trends and issues of e-learning. Procedia - Social and Behavioral Sciences, 77(Supplement C):429 – 442, 2013. The Harmony of Civilization and Prosperity for All: Selected Papers of Beijing Forum (2009-2010).

[2] M. Yamamoto, T. Sekiya, and K. Yamaguchi. Relationship between programming concepts underlying programming skills. In 2011 International Conference on Information Technology Based Higher Education and Training, pages 1–7, Aug 2011.

[3] Herbert Schildt. Teach Yourself C. McGraw-Hill Osborne Media, 1997.

[4] C tutor. http://www.pythontutor.com/c.html.

[5] Dusan Jovanovic and Slobodan Jovanovic. An adaptive e-learning system for java programming course, based on dokeos le. Computer Applications in Engineering Education, 23(3):337–343, 2015.

[6] Iterative information presentation learning system (proposed learning system in this paper). free to use, register questions and answer the questions. https://www.iipls.net.

# 第 8 章

# 結論

## 8.1 研究成果のまとめ

本研究では，経験に着目した学習教材について理論面と実践面の双方で調査を行い，経験に着目した一貫性のある学習理論と方法の提案，評価を行った．これにより，従来は経験型学習で行えていない学習形式と出題内容の理論的な裏付けをまとめ学習方法として提案を行い，それを踏まえた学習システムの制作とと評価も行うことで，学習理論と方法の提案，評価を一貫して行った．

結果として以下の特徴を持つ学習方法を提案した．

- 学習形式の特徴（素朴理論，熟達化，動機づけ知見）
  - 学習要素の説明は問題に取り掛かれる範囲まで（それ以上は行わない）
  - 学習問題の正誤即時判定 (誤回答時は正答を表示)
  - 読み学習の連続出題
    - 出力結果の選択形式 (三択)
    - 文字ベースの教材
- 出題の特徴（スキーマ，動機づけ知見）
  - 出題内容は中心問題と類似問題の交互出題
    - 類似問題の変更点:出力→他連接処理→反復や分岐条件
  - 誤回答選択肢の明確化
- その他の特徴（同時並列に機能する複数の能力）
  - 理解型教材と経験型教材（上記の学習形式と出題の特徴の教材）の併用

この学習方法の結果について，従来型の手法である，代表的な教科書 [1] の解説部分と，デバッガ形式の Visualize ツール [2] と書き課題の組み合わせ学習との比較において以下の改善が示された．

「要素会得の課題」について，実験５で読みスキルの改善，実験６において書きスキルの改善が見られた．「理解多様性の課題」について，実験６で低得点群の得点底上げ傾向があり，改善が見られた．「動機づけの課題」について，実験５，実験６で学習者の離脱と見られる活動が見られなかった．一方で，比較した従来型の手法では離脱と見られる活動が一部見られた．さらに，提案手法では学習中の理解度把握が容易である特徴が実験５と実験６で示されたため，学習中のフォローについても可能な拡張性が示された．「知識応用の課題」について，実験２において，豊かなスキーマを構築している学習者は，知らないとするコードトレースが可能であったことから，スキーマの考慮が重要であるという理論的な確認が行えた．実践的な確認としては十分に示されておらず，実験６の条件では提案手法の改善が見られなかった（従来手法と比較して差が見られなかった）．

## 8.2 今後の研究課題，展望 – 提案システムの公開と改善

本手法の課題として次の点が挙げられる．

- 多人数，長期的な実験調査による応用力を含む能力向上と定着率の確認
- 反復出題の詳細な方法調査
- 理解型教材と経験型教材の組み合わせ方詳細調査

本研究では，限定した要素の学習に特化して一時間程度の範囲で学習できる内容について調査を行った．この点について，応用力の向上を意図した学習要素を含めており，実験２において理論的には確認が出来たものの，その学習効果の確認までは行えなかった．また，実験５においてはトレーススキルの向上が見られたが，実験６では見られなかった．この点は出題内容にも起因することが考えられるため，異なる環境で再度確認が必要である．そこで，課題としては一般的な学習コースである１ヶ月や数ヶ月の期間のまとまった内容でも同様の結果が見られるかどうか，応用力が向上するかどうか，また長時間経過した後の定着率はどうであるかについて検証が必要である．

また，デフォルト値とスロットの獲得を目的とした出題である中心問題の反復出題であるが，これについては，中心問題とその他の問題を交互に出題する手法をとった．しかしなが

ら，これについては他の組み合わせ方も考えられる．例としては中心問題の後に 2 問や 3 問連続で中心問題の類似問題を出題する方式である．これらの中でどれが最適であるかについては調査の余地がある．同様に，理解型教材と経験型教材の組み合わせ方についても，調査の余地がある．

本研究にて提案した手法についてはシステムにまとめ，一般公開を行った [3]．これは，指導者側が問題の設定を行い，学習者側が回答を行う事ができる，出題と回答の双方の機能を有しており，使用については，特別な制限を設けずに無料で行えるものである．また，授業でも使用が可能なように，問題作成時に隠し公開（特定の URL を知っているもののみが回答可能）も可能である．また，特徴としては，中心問題を設定して類似問題と交互に出題する点については，自動で出題順を設定する機能を設けたため，少ない手間で問題作成が可能になる．さらに，出題順にについては調整が可能であるため，課題として示した交互に出題する方式の設定も可能となる．

今後の展望については，本システムの公開を続け，提案する学習方法を加味した学習ツールの提供を通じて，当該分野への学習改善に貢献するとともに，各箇所で発表するなどして認知度を向上させ，学習結果の蓄積と評価を行い，改善点の発見とその対応を行う．

本研究は経験により得られる能力に着目したプログラミング学習方法を提案した．提案手法は経験を重視する学習教材において，従来は一貫した理論から方法がなかった課題に対応し，理論的に裏付けのある学習方法を用い，検証された学習方法や出題内容の確立を行った．このプログラミング学習方法を実践することにより，大学生を対象とした入門レベルのプログラミング読みと書きスキルの向上に加え，低得点群の底上げと脱落の防止効果も認められた．すなわち，本研究の成果は日本のみならず，諸外国のプログラミング教育に大きなインパクトを与えるものである．本研究の成果が活用され，世界中に優秀なプログラマーが増える事を願ってやまない．

# 参考文献

[1] Herbert Schildt. Teach Yourself C. McGraw-Hill Osborne Media, 1997.

[2] C tutor. http://www.pythontutor.com/c.html.

[3] Iterative information presentation learning system (proposed learning system in this paper). free to use, register questions and answer the questions. https://www.iipls.net.

# 論文目録

## 学術論文

1. Satoru Kikuchi and Kazuhiko Hamamoto. The effects of a schema on applied programming skills in an introductory class. IEEJ transactions on electronics, information and systems, 136(7):9951000, 2016.（本論文は実験２と関連している）
2. Satoru Kikuchi and Kazuhiko Hamamoto. Investigating the relationship between tracing skill and modification skill for different programming statements. Technical Report 1, THE SCHOOL OF INFORMATION AND TELECOMMUNICATION ENGINEERING, TOKAI UNIVERSITY, September 2016.（本論文は実験３と関連している）
3. Satoru Kikuchi and Kazuhiko Hamamoto. Implementation and Pilot Evaluation of an Iterative Learning Environment for Programming Education. 2018 International Conference on Learning and Teaching in Computing and Engineering (LaTICE) (accepted as a poster)（本論文は 実験６と関連している）

# 実験１に使用した問題

知識課題の問題は Fig. 8.1 に示す.


Fig. 8.1　Fill in a blank tasks for Experiment1.

瞬間判断課題の問題は以下になる．

```
int value = 100;
```

```
float price=1000, tax = 0.8;
```

```
int 100;
```

```
int price=1000, float tax = 0.8;
```

```
float (int)value =10.1;
```

```
float value =(int)10.1;
```

```
int array[30];
array[10] = 10;
```

```
int array[30];
array[5.5] = 10;
```

```
int array[30];
array[100] = 10;
```

```
int abc = 10;
200 = abc;
```

```
int array[30];
150 = array[10];
```

```
int 100 = abc;
```

```
int 100 = 100;
```

```
if(1) printf("abc");
```

```
if(int i=10) printf("abc");
```

```
int abc =10;
if(abc= 100 < 200) printf("abc");
```

```
if(100==100) printf("abc");
```

```
int i;
for(i=0;i<100;i++) printf("abc");
```

```
int i;
for(i=100;i>=0;i--) printf("abc");
```

```
int i;
for(i=0;i<100) printf("abc");
```

```
int i;
for(i<100;i=10) printf("abc");
```

```
int i;
for(i<100;i=10;i--) printf("abc");
```

```
int i;
for(i=10;i<100;i--) printf("abc");
```

```
int i;
for(i=10;i<100;i=i+2) printf("abc");
```

```
int i;
for(i=10;i<100;i=i) printf("abc");
```

柔軟性課題の問題は以下になる．

```
// 条件：for 文を１回使うこと
// 条件：printf("[abc]"); を使うこと
// 出力結果：[abc][abc][abc][abc][abc]
#include<stdio.h>
int main(){
    // ここに回答を記入上下の行は固定．
    return 0;
}
```

```
// 条件：for 文を使用しないこと
// 出力結果：[abc][abc][abc][abc][abc]
#include<stdio.h>
int main(){
    // ここに回答を記入上下の行は固定．
    return 0;
}
```

```
// 条件：for 文を使用しないこと
// 条件：printf 文を回使用すること3
// 出力結果：[abc][abc][abc][abc][abc]
#include<stdio.h>
int main(){
    // ここに回答を記入上下の行は固定．
    return 0;
}
```

```
// 条件：1 から10 までの整数の総和を計算し表示する
// 条件：計算にfor 文を使用すること
// 条件：変数宣言は int a,b,c,d,e; のみとする．（これ以外の変数の使用は禁止とする）
// 出力結果：55
#include<stdio.h>
int main(){
    int a,b,c,d,e;
    // ここに回答を記入上下の行は固定．
    return 0;
}
```

```
// 条件：1 から11 までの奇数の総和を計算し表示する
// 条件：計算にfor 文を使用すること
// 条件：変数宣言は int a[5]; のみとする．（これ以外の変数の使用は禁止とする）
// 出力結果：
#include<stdio.h>
```

```
int main(){
    int a[5];
    // ここに回答を記入上下の行は固定.
    return 0;
}
```

記憶課題の問題は以下になる.

```
#include<stdio.h>
int main(){
    for(int i=1930;i<=2014;i++){
        printf("%年d\n",i);}

    return 0;
}
```

```
#include<stdio.h>
int main(){
    int start = 1930, end = 2014;
    for(int i=start;i<=end;i++){
        printf("%年d\n",i);
    }
    return 0;
}
```

```
#include<stdio.h>
int main(){
    int start = 1930, end = 2014;
    for(int i=start;i<=end;i++){
        if((i-start)%4==0){
            printf("%年d FIFA ワールドカップ開催\n",i);
        }
    }
    return 0;
}
```

```
#include<stdio.h>
int main(){
    int start = 1930, end = 2014;
    for(int i=start;i<=end;i++){
        if((i-start)%4==0){
            if(i==1942 || i==1946){
                printf("%年d 戦争により中止\n",i);
            } else {
                printf("%年d FIFA ワールドカップ開催\n",i);
            }
        }
    }
    return 0;
}
```

```
#include<stdio.h>
int main(){
    int start = 1930, end = 2014, count = 0;
    for(int i=start;i<=end;i++){
        if((i-start)%4==0){
            if(i==1942 || i==1946){
                printf("%年  戦争により中止d\n",i);
            } else {
                count += 1;
                printf("%年d 第%回d FIFA ワールドカップ\n",i, count);
            }
        }
```

```
    }
    return 0;
}
```

アルゴリズム説明の資料は Fig. 8.2 に示す．なお，この説明画像の出典は以下になる．

出典：「バケットソート」『フリー百科事典　ウィキペディア日本語版』 2014 年 7 月 11 日 (金) 12:10　UTC URL:http://ja.wikipedia.org/wiki/%E3%83%90%E3%82%B1%E3%83%83%E3%83%88%E3%82%BD%E3%83%BC%E3%83%88

アルゴリズム理解課題の問題は Fig. 8.3 に示す．

アルゴリズム実装課題の問題は Fig. 8.4 に示す．

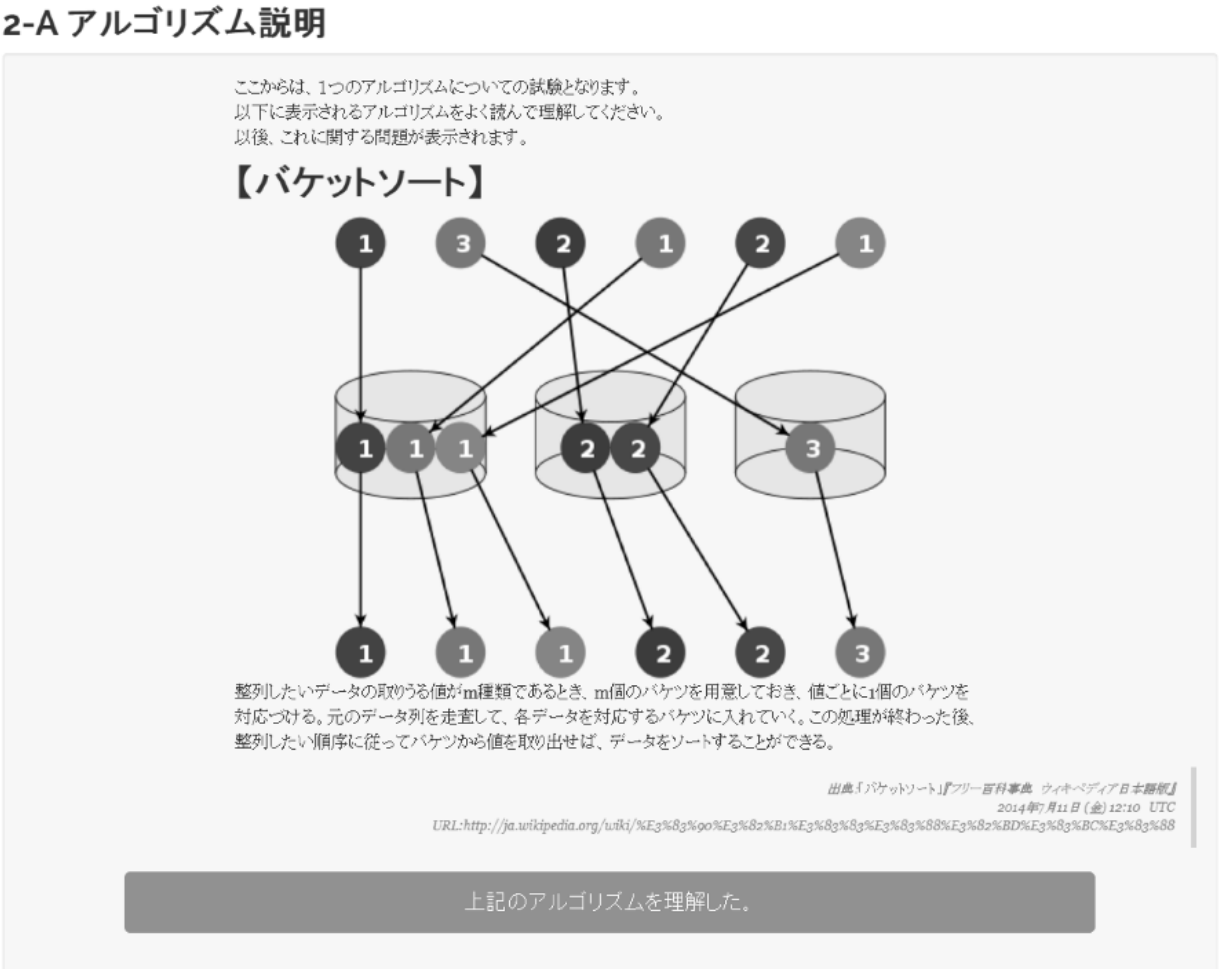


Fig. 8.2　Algorithm leaning task for Experiment1.

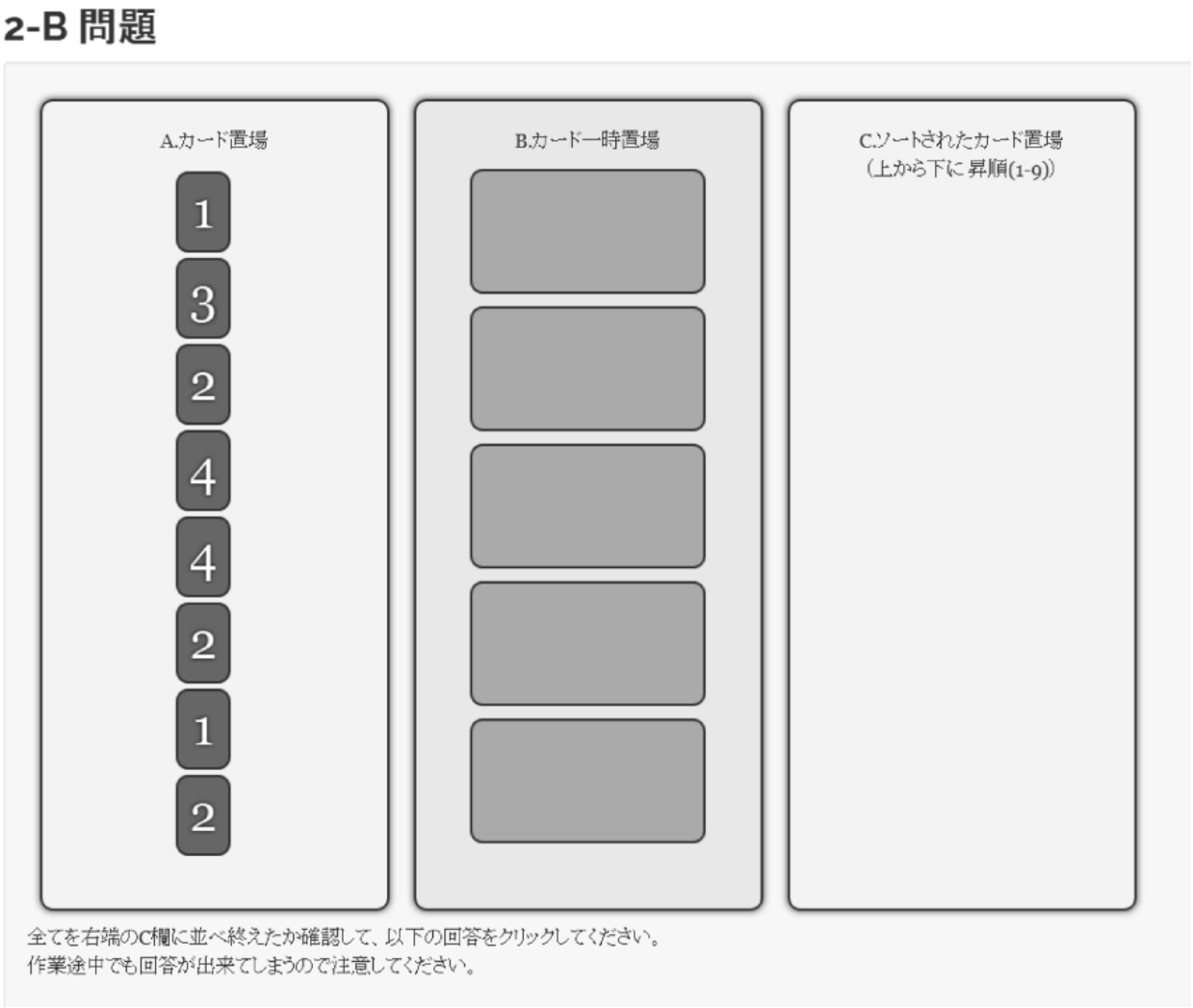


Fig. 8.3　Algorithm card task for Experiment1.

**2-C 問題** 以下の欄を入力しバケットソートを用いて配列arrの内容を並び替え配列arrAnsに代入し、実行結果の通り昇順(1-9)で表示せよ。

※キーボード操作はすべて記録する。記録の関係上、すべて半角モードで入力すること。全角文字、日本語などは使用禁止。
※安定ソートでなくともよい(実行後に、実行結果と同じ表示になればよい)

回答欄

```
#include <stdio.h>
int main(){
        int arr[8] = {1,3,2,4,4,2,1,2};
        int bucket[5] = {0,0,0,0,0};
        int arrAns[8] = {0,0,0,0,0,0,0,0};
        // 変数は自由に宣言してよい
}
```

実行結果

1,1,2,2,2,3,4,4,

Fig. 8.4 Algorithm implementation task for Experiment1.

# 実験２に使用した問題

本実験は，問題用紙を被験者に配布して試験を行った．その内容は 8.5–8.11 に示す．

プログラミングの経験量が開発力に与える影響に関する研究　実験 2015年5月

ID:　　1 / 7

## アンケート

質問の回答を回答欄に記載または選択して記入せよ。

~~1.年齢~~

~~2.性別　□男性　□女性~~

3.プログラミング入門の成績
※右の点線内の選択肢から記載

4.プログラミング応用の成績
※右の点線内の選択肢から記載

・未履修
・不合格
・S
・A
・B
・C

5.授業時間以外でプログラミングを行うか
（頻度は週1回程度が基準）
□はい　□いいえ

## 問1 ソースコード記述課題

実行結果となる処理を「ソースコード」欄に記載せよ。
また、記述したコードに関する自らの経験について、該当するものを選択肢より選択せよ。

■「ソースコード」の記載について補足

・言語はC言語で記載すること（本試験は全課題通じて、ANSI C89準拠コンパイラでの実行結果を基準とする）
・実行可能な処理で各問題の指示を満たす出力であれば、変数宣言、分岐、反復など、どのような処理を記載しても良い。
・全ての出力は、**変数の値のみを出力**すること。（「文字/数字」のみの出力は禁止である。ただし、カンマ","は例外とする）
・予め記載されている処理を変更してはならない
・実行結果と変数の初期値は、一つの例であり、仮に異なる初期値が変数に入っていたとしても、各問題の指示通りとなる処理を記載すること。

■「記載した処理の経験」について補足

・ソースコードで自覚している経験を回答せよ。
・回答は下記の問題の経験表を参考に、それぞれの回答欄に該当するものに○印をつけよ。
・過去の経験で，変数名や値は異なっても同等のソースコード の経験があると感じれば、ありと回答せよ。
・過去に経験した 大きなソースコードの一部分として、各問題で回答した処理を経験 した場合も、経験ありと回答せよ。

問題の経験表

| |
|---|
| なし：経験無し（過去に見た/書いた事のない処理） |
| 見　：見たことはあるが、自ら書いたことはない |
| 書少：自ら書いた経験が数回あると感じる |
| 書多：自ら書いた経験が10回以上あると感じる |

■問1 － 1

a,b,cの値で最大の値を出力する処理を記載せよ。
※変数の初期値が変わっても、最大の値を出力する処理を記載せよ。同じ値が存在する場合は、どれか1つを出力すること。

実行結果 30

記載した処理の経験　↓該当に○

| ソースコード | なし | 見 | 書少 | 書多 |
|---|---|---|---|---|

```
#include<stdio.h>
int main(){
    int a=10, b=20, c=30;

    return 0;
}
```

■問1 － 2

a,b,cの値で2番目に大きい値を出力する処理を記載せよ。
※変数の初期値が変わっても、2番目に大きい値を出力する処理を記載せよ。a,b,cの値は、必ず異なる値が入っているものとする。

実行結果 20

記載した処理の経験　↓該当に○

| ソースコード | なし | 見 | 書少 | 書多 |
|---|---|---|---|---|

```
#include<stdio.h>
int main(){
    int a=10, b=20, c=30;

    return 0;
}
```

Fig. 8.5　Problem paper 1 of 7 in experiment 2.

プログラミングの経験量が開発力に与える影響に関する研究 実験 2015年5月

ID: 2 / 7

■問1 － 3

配列arrの要素を順に出力する処理を記載せよ。
※変数の初期値や定数Nが変わっても、要素の値を順に出力する処理を記載せよ。なお、Nが変わった場合は、配列の全ての要素に適当な整数が入るものとする。

実行結果 12,15,14,20,10,

記載した処理の経験 ↓該当に○

| ソースコード | なし | 見 | 書少 | 書多 |
|---|---|---|---|---|

```
#include<stdio.h>
#define N 5
int main(){
    int arr[N] = {12,15,14,20,10};




    return 0;
}
```

■問1 － 4

配列arrの要素を次の順番で出力する処理を記載せよ。
2,1,4,3,6,5,8,7,10,9,12,11・・・（数値は要素の番号）
※変数の初期値や定数Nが変わっても、要素の値を指定の順に出力する処理を記載せよ。なお、Nが変わった場合は、配列の全ての要素に適当な整数が入るものとする。

実行結果 15,12,20,14,10,

記載した処理の経験 ↓該当に○

| ソースコード | なし | 見 | 書少 | 書多 |
|---|---|---|---|---|

```
#include<stdio.h>
int main(){
#define N 5
int main(){
    int arr[N] = {12,15,14,20,10};




    return 0;
}
```

■問1 － 5

配列arrの要素の値の総和を出力する処理を記載せよ。
※変数の初期値が変わっても、要素の値の総和を出力する処理を記載せよ。なお、Nが変わった場合は、配列の全ての要素に適当な整数が入るものとする。

実行結果 71

記載した処理の経験 ↓該当に○

| ソースコード | なし | 見 | 書少 | 書多 |
|---|---|---|---|---|

```
#include<stdio.h>
#define N 5
int main(){
    int arr[N] = {12,15,14,20,10};




    return 0;
}
```

■問1 － 6

配列arrの要素で次の計算を行い出力する処理を記載せよ。
1*2 + 2*3 + 3*4 + 4*5 + 5*1 （数値は要素の番号）
※以下のソースコードの初期値 {2,4,3,1,5} の場合の計算は
2*4 + 4*3 + 3*1 + 1*5 + 5*2 = 8+12+3+5+10 = 38
※変数の初期値が変わっても、指定の計算結果を出力する処理を記載せよ。なお、Nが変わった場合は、配列の全ての要素に適当な整数が入るものとする。

実行結果 38

記載した処理の経験 ↓該当に○

| ソースコード | なし | 見 | 書少 | 書多 |
|---|---|---|---|---|

```
#include<stdio.h>
#define N 5
int main(){
    int arr[N] = {2,4,3,1,5};




    return 0;
}
```

Fig. 8.6 Problem paper 2 of 7 in experiment 2.

プログラミングの経験量が開発力に与える影響に関する研究　実験 2015年5月

ID:　　3 / 7

■問1 － 7

配列arrの要素で最大の値を出力する処理を記載せよ。
※変数の初期値が変わっても、要素の最大の値を出力する処理を記載せよ。なお、Nが変わった場合は、配列の全ての要素に適当な整数が入るものとする。

実行結果 20

記載した処理の経験　↓該当に○

| ソースコード | なし | 見 | 書少 | 書多 |
|---|---|---|---|---|

```
#include<stdio.h>
#define N 5
int main(){
    int arr[N] = {12,15,14,20,10};

    return 0;
}
```

Fig. 8.7　Problem paper 3 of 7 in experiment 2.


Fig. 8.8 Problem paper 4 of 7 in experiment 2.

プログラミングの経験量が開発力に与える影響に関する研究　実験 2015年5月

ID:　　5 / 7

■問2 − 12

```
1 int arr[10];
2 arr[1] = 13;
3 printf("%d",arr[1]);
```

実行結果　正常　エラー　終らず
出力結果
経験　なし　見　書少　書多

■問2 − 13

```
1 int a=20;
2 a++;a--;a++;
3 printf("%d",a);
```

実行結果　正常　エラー　終らず
出力結果
経験　なし　見　書少　書多

■問2 − 14

```
1 int a=21;
2 printf("%d",a++);
3
```

実行結果　正常　エラー　終らず
出力結果
経験　なし　見　書少　書多

■問2 − 15

```
1 int a=11;
2 a = 1;;;;;
3 printf("%d",a);
```

実行結果　正常　エラー　終らず
出力結果
経験　なし　見　書少　書多

■問2 − 16

```
1 int a=12;
2 10 = a;
3 printf("%d",a);
```

実行結果　正常　エラー　終らず
出力結果
経験　なし　見　書少　書多

■問2 − 17

```
1 int a=13;
2 =a;
3 printf("%d",a);
```

実行結果　正常　エラー　終らず
出力結果
経験　なし　見　書少　書多

■問2 − 18

```
1 int a=15;
2 a += 1;
3 printf("%d",a);
```

実行結果　正常　エラー　終らず
出力結果
経験　なし　見　書少　書多

■問2 − 19

```
1 int a=16;
2 a.a = 20;
3 printf("%d",a);
```

実行結果　正常　エラー　終らず
出力結果
経験　なし　見　書少　書多

■問2 − 20

```
1 int a=17;
2 a++;
3 printf("%d",a);
```

実行結果　正常　エラー　終らず
出力結果
経験　なし　見　書少　書多

■問2 − 21

```
1 int a=18;
2 a+5;
3 printf("%d",a);
```

実行結果　正常　エラー　終らず
出力結果
経験　なし　見　書少　書多

■問2 − 22

```
1 int a=19;
2 a==10;
3 printf("%d",a);
```

実行結果　正常　エラー　終らず
出力結果
経験　なし　見　書少　書多

■問2 − 23

```
1 int a=10, b=6;
2 a = b = 20;
3 printf("%d, %d",a, b);
```

実行結果　正常　エラー　終らず
出力結果
経験　なし　見　書少　書多

■問2 − 24

```
1 int a=10, b=5;
2 a = a<=b;
3 printf("%d, %d",a, b);
```

実行結果　正常　エラー　終らず
出力結果
経験　なし　見　書少　書多

■問2 − 25

```
1 int a=20, b=20;
2 if(a==b) printf("a==b");
3 if(a>=b) printf("a>=b");
4 if(a<b) printf("a<b");
```

実行結果　正常　エラー　終らず
出力結果
経験　なし　見　書少　書多

■問2 − 26

```
1 int a=10, b=21;
2 if(a!=b)
3   printf("a!=b");
```

実行結果　正常　エラー　終らず
出力結果
経験　なし　見　書少　書多

■問2 − 27

```
1 int a=9, b=20;
2 if(a>b){
3   printf("a");
4 } else {
5   printf("b");
6 }
```

実行結果　正常　エラー　終らず
出力結果
経験　なし　見　書少　書多

■問2 − 28

```
1 int a=10, b=22;
2 if(a!b)
3   printf("a!b");
```

実行結果　正常　エラー　終らず
出力結果
経験　なし　見　書少　書多

■問2 − 29

```
1 if(1){
2   printf("abc");
3 }
```

実行結果　正常　エラー　終らず
出力結果
経験　なし　見　書少　書多

■問2 − 30

```
1 int a=11, b=20;
2 if(a>b)  a=30; printf("%d",a);
3
```

実行結果　正常　エラー　終らず
出力結果
経験　なし　見　書少　書多

■問2 − 31

```
1 int a=10, b=20;
2 if(a>b){
3   printf("a>b");
4 } else {
5   printf("other");
6 } else if(a<b) {
7   printf("a<b");
8 }
```

実行結果　正常　エラー　終らず
出力結果
経験　なし　見　書少　書多

Fig. 8.9　Problem paper 5 of 7 in experiment 2.

プログラミングの経験量が開発力に与える影響に関する研究　実験 2015年5月

ID:　　6 / 7

■問2 － 32

```
1 int i=0;
2 for(i=0;i<3; i++)
3   printf("a");
```

実行結果　正常　エラー　終らず
出力結果
経験　なし　見　書少　書多

■問2 － 33

```
1 int i=0;
2 for(i=0;i<5; i=i-1)
3   printf("a");
```

実行結果　正常　エラー　終らず
出力結果
経験　なし　見　書少　書多

■問2 － 34

```
1 int i=0;
2 for(i=0;i<4; i=i+2)
3   printf("a");
```

実行結果　正常　エラー　終らず
出力結果
経験　なし　見　書少　書多

■問2 － 35

```
1 int i=0;
2 for(;;){
3   printf("a");
4 }
```

実行結果　正常　エラー　終らず
出力結果
経験　なし　見　書少　書多

■問2 － 36

```
1 int i=0;
2 for(i=0;i==5; i=i+2)
3   printf("a");
```

実行結果　正常　エラー　終らず
出力結果
経験　なし　見　書少　書多

■問2 － 37

```
1 int i=0;
2 for(i=0;i<3; i++){
3   printf("%d",i);
4 }
```

実行結果　正常　エラー　終らず
出力結果
経験　なし　見　書少　書多

■問2 － 38

```
1 int i=0;
2 for(;i<3; i=i+2)
3   printf("a");
```

実行結果　正常　エラー　終らず
出力結果
経験　なし　見　書少　書多

■問2 －　39

```
1 int i=0, a=0;
2 for(i=0;a<3;i++ ){
3   printf("%d",i);
4   a = a+2;
5 }
```

実行結果　正常　エラー　終らず
出力結果
経験　なし　見　書少　書多

■問2 － 40

```
1 int j=0;
2 for(;j<3; )
3   printf("a");
4   j=j+1;
```

実行結果　正常　エラー　終らず
出力結果
経験　なし　見　書少　書多

■問2 － 41

```
1 int arr[3] = {2,5,4}, i;
2 for(i=0;i<3; i++){
3   printf("%d,",arr[i]);
4 }
```

実行結果　正常　エラー　終らず
出力結果
経験　なし　見　書少　書多

■問2 － 42

```
1 int arr[3] = {1,4,3}, a=0, i;
2 for(i=0;i<3; i++){
3   a = arr[i];
4 }
5 printf("%d",a);
```

実行結果　正常　エラー　終らず
出力結果
経験　なし　見　書少　書多

■問2 － 43

```
1 int arr[3] = {2,5,3}, a=0, i;
2 for(i=1;i<3; i++){
3   if(arr[a] < arr[i]){
4     a = i;
5   }
6 }
7 printf("%d",arr[a]);
```

実行結果　正常　エラー　終らず
出力結果
経験　なし　見　書少　書多

※以下の問題44,45 は、ソースコードの省略を行わず全ての行が記載されている。

■問2 － 44

```
1 #include<stdio.h>
2 #define N 3
3 int main(){
4   int arr[N] = {4,3,5}, i;
5   for(i=0;i<N;i++){
6     printf("%d,",arr[i]);
7   }
8 }
```

実行結果　正常　エラー　終らず
出力結果
経験　なし　見　書少　書多

■問2 － 45

```
1 #include<stdio.h>
2 #define N 3
3 int main(){
4   int arr[N] = {4,3,5}, i;
5   for(i=0;i<N;i++){
6     printf("%d,",N);
7   }
8 }
```

実行結果　正常　エラー　終らず
出力結果
経験　なし　見　書少　書多

Fig. 8.10　Problem paper 6 of 7 in experiment 2.

プログラミングの経験量が開発力に与える影響に関する研究　実験 2015年5月

ID:　　7 / 7

## 問3 ソースコード記述課題

実行結果となる処理を「ソースコード」欄に記載せよ。
また、記述したコードに関する自らの経験について、該当するものを選択肢より選択せよ。

■「ソースコード」の記載について補足

・言語はC言語で記載すること（本試験は全課題通じて、ANSI C89準拠コンパイラでの実行結果を基準とする）
・実行可能な処理で各問題の指示を満たす出力であれば、変数宣言、分岐、反復など、どのような処理を記載しても良い。
・全ての出力は、**変数の値のみを出力**すること。（「文字/数字」のみの出力は禁止である。ただし、カンマ","は例外とする）
・予め記載されている処理を変更してはならない
・実行結果と変数の初期値は、一つの例であり、仮に異なる初期値が変数に入っていたとしても、各問題の指示通りとなる処理を記載すること。

■「記載した処理の経験」について補足

・ソースコードで自覚している経験を回答せよ。
・回答は下記の問題の経験表を参考に、それぞれの回答欄に該当するものに〇印をつけよ。
・過去の経験で，変数名や値は異なっても同等のソースコード の経験があると感じれば、ありと回答せよ。
・過去に経験した 大きなソースコードの一部分として、各問題で回答した処理を経験 した場合も、経験ありと回答せよ。

問題の経験表

| |
|---|
| なし：経験無し（過去に見た/書いた事のない処理） |
| 見　：見たことはあるが、自ら書いたことはない |
| 書少：自ら書いた経験が数回あると感じる |
| 書多：自ら書いた経験が10回以上あると感じる |

■問3 － 1

配列arrの要素で最少の値を出力する処理を記載せよ。
※変数の初期値が変わっても、要素の最少の値を出力する処理を記載せよ。なお、Nが変わった場合は、配列の全ての要素に適当な整数が入るものとする。

実行結果 10

記載した処理の経験　↓該当に〇

| ソースコード | なし | 見 | 書少 | 書多 |
|---|---|---|---|---|

```
#include<stdio.h>
#define N 5
int main(){
    int arr[N] = {12,15,14,20,10};

    return 0;
}
```

Fig. 8.11　Problem paper 7 of 7 in experiment 2.

# 実験３に使用した問題

以下，トレースタスクの問題である．共通的な，include 文や return 文は省略されている．

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);
```

以下，修正タスクの問題である．共通的なコードである，include 文や return 文は省略されている．

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);
```

# 実験４に使用した問題

以下は，文章題の問題である．問題文に続いて，回答の選択肢を４つ表示する（※がついている選択肢が正答である）．

---

question:Haskell 言語のパラダイムはどれが正しいか？
ans1:命令型
ans2:関数型　※
ans3:オブジェクト指向
ans4:プロトタイプ指向

---

question:Haskell の特徴は，次のうちどれが適切か？
ans1:中間言語
ans2:ポリモフィズム
ans3:メモリ操作
ans4:遅延評価　※

---

question:Haskell で，max 関数を呼び出す際の書式は次のうちどれが正しいか？
ans1:max (10 5)
ans2:max (10, 5)
ans3:max 10 5　※
ans4:max 10, 5

---

question:succ 9 の結果は，次のうちどれが正しいか？
ans1:0
ans2:9
ans3:10　※
ans4:18

---

question:10 `div` 4 のような書き方の関数をなんと呼ぶか？
ans1:中置関数　※
ans2:中間関数
ans3:演算関数
ans4:拡張関数

---

question:関数の命名で正しくないものを選べ
ans1:bigNumber
ans2:BigNumber　※
ans3:bignumber'
ans4:bignumber999

---

question:関数名にアポストロフィ (') を使用するのは一般的にどのような場合か？
ans1:使用できない
ans2:意味は変わらない
ans3:逆関数の作成
ans4:少し変更したバージョン　※

---

question:関数名が大文字から始まるのは一般的にどのような場合か？
ans1:使用できない　※
ans2:意味は変わらない
ans3:逆関数の作成
ans4:少し変更したバージョン

---

question:リストの添え字はいくつから始まるか？
ans1:-1
ans2:0　※
ans3:1
ans4:10

---

question:リストの要素は，様々な型を含めることができるか？
ans1:自由にすべての型を混在できる
ans2:すべて同じ型が必須　※
ans3:数値や文字で統一されれば混在可能
ans4:要素が２つまでならば混在可能

---

question:リスト範囲外のアクセスはどのような結果となるか？
ans1:エラー　※
ans2:0 が返る
ans3:[] が返る
ans4:NULL が返る

---

question:内包表記の条件は他になんと呼ぶか？
ans1:選択
ans2:分岐
ans3:述語　※
ans4:主語

---

question:ペアと実質的に同じものは何か
ans1:要素が２つのリスト
ans2:要素が２つのタプル　※
ans3:引数が２つの関数
ans4:同じ結果となる２つの関数

---

question:タプルの要素は，様々な型を含めることができるか？
ans1:自由にすべての型を混在できる　※
ans2:すべて同じ型が必須
ans3:数値や文字で統一されれば混在可能
ans4:要素が２つまでならば混在可能

---

question:長さの異なるリストの zip はどのような結果となるか？
ans1:エラー
ans2:短いリストの分だけ処理される　※
ans3:不足分は 0 となり，長いリストの分だけ処理される．
ans4:不足分は NULL となり，長いリストの分だけ処理される．

---

以下は，トレースタスクの問題となる．

```
main = do {
    print ( 2000 - 1950 );
}
```

```
main = do {
    print ( 5 + "11ama" );
}
```

```
main = do {
    print ( min (9 10) );
}
```

```
main = do {
    print ( \`div\` 92 10 );
}
```

```
main = do {
    print ( min 5 4 * 5  );
}
```

```
tripleMe x = x + x + x
main = do {
    print ( tripleMe 50 );
}
```

```
MrHaskell = \"It's a-me, Mr.Haskell!\"
main = do {
 print ( MrHaskell )
}
```

```
doubleUs x y = x * 2 + y * 2
main = do {
    print ( doubleUs 4 9 );
}
```

```
main = do {
    print ( [3,4] ++ [11,12] ++ [20,21] );
}
```

```
main = do {
    print ( [1,2,3,4] + [5,5,5,5] );
}"
```

```
main = do {
    print ( \"Steve Jobs\" : 6 );
}
```

```
b = [[1,2,3,4], [5,3,3,3], [1,2,2,3,4], [1,2,3] ]
main = do {
    print ( b !! 2 );
}
```

```
main = do {
    print ( head [6,3,8,9,1] );
}
```

```
main = do {
    print ( head [] );
}
```

```
main = do {
    print ( take 2 [5,4,3,2,1] );
}
```

```
main = do {
    print ( 4 \`elem\` [3,4,5,6] );
}
```

```
main = do {
    print ( ['a'....'z'] );
}
```

```
main = do {
    print ( [2,4..10] );
}
```

```
main = do {
    print ( take 10 (repeat 5) );
}
```

```
main = do {
    print ( [x*3 | x <- [1..5], x / 2 == 1] );
}
```

```
main = do {
    print ( [x*2 | x <- [1..10]] );
}
```

```
removeLetter st = [ c | c <- st, c \`elem\` ['A'.. 'Z'] ]
main = do {
    print ( removeLetter \"Hahaha! Nahaha!\" );
}
```

```
main = do {
    print ( fst (8,11) );
}
```

```
main = do {
    print ( zip [1,2,3] [9,9,9] );
}
```

以下は，書きタスクの問題である．

```
// 出力結果：30
// 条件：下部の参考ソースの main do = {} の括弧の中は，変更せず用いること．
// 以下，サンプルコード
doubleUs x y = x
main = do {
    print ( doubleUs 5 10 );
}
```

```
// 出力結果：[(1,"one"),(2,"two"),(3,"three"),(4,"four"),(5,"five"),(6,"six")]
// 条件：下部の参考ソースの zip を用いること．
// 以下，サンプルコード
main = do {
    print ( zip [1..5] ["one", "two", "three", "four", "five"]  );
}
```

```
// 出力結果：30
// 条件：引数を一つもち，渡された値の３倍を返す関数 sanbai を作成して使用せよ．
```

# 実験５に使用した問題

以下，記憶タスクの問題である．

```
#include<stdio.h>
int main () {
    int a = 0;
    int b = 0;
    int i = 0;
    int arr[7] = {10, 0, -10, 5, 5, 9999, 10};
    for(i=0;i<=6;i=i+1){
        if(arr[i] == 9999){
            break;
        }
        if(arr[i] < 0){
            arr[i] = 0;
        }
        a = a + arr[i];
        b = b + 1;
    }
    printf("value␣is␣%d\n", a / b);
    return 0;
}
```

以下，トレースタスクの問題である．共通的な include 文と return 文は省略されている．

```
    int a = 10;
    int b = 8;
    int c = 5;
    if (a + b > 15) {
        printf("ABC\n");
    }

    if (b > a && b > c) {
        printf("DEF\n");
    }

    if (a == b || b + 2 == a) {
        printf("GHI\n");
    }
```

```
    int i = 0;
    for(i=0;i<=2;i=i+1){
        printf("i␣is␣%d\n", i);
    }
```

```
    int a = 0;
    int i = 0;
    int arr[5] = {2,5,3,8,2};
    for(i=0;i<=4;i=i+1){
        a = a + arr[i];
```

```
    }
    printf("%d\n", a);
```

```
    int a = 0;
    int i = 0;
    int arr[5] = {2,5,3,8,2};
    for(i=0;i<=4;i=i+1){
        if(arr[i] > a){
            a = arr[i];
        }
    }
    printf("%d\n", a);
```

```
    int i = 0;
    int j = 0;
    for(i=0;i<=1;i=i+1){
        for(j=0;j<=1;j=j+1){
            printf("i=%d,j=%d\n", i, j);
        }
    }
```

```
    int a = 0;
    int i = 0;
    int arr[3] = {9,4,3};
    for(i=0;i<=1;i=i+1){
        if (arr[i] > arr[i+1]){
            a = arr[i];
            arr[i] = arr[i+1];
            arr[i+1] = a;
        }
    }
    for(i=0;i<=2;i=i+1){
        printf("arr[%d]=%d\n",i, arr[i]);
    }
```

```
    int a = 0;
    int i = 0;
    int j = 0;
    int arr[5] = {9,4,3,7,1};
    for(i=0;i<=3;i=i+1){
        for(j=0;j<=3;j=j+1){
            if (arr[j] > arr[j+1]){
                a = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = a;
            }
        }
    }
    for(i=0;i<=4;i=i+1){
        printf("arr[%d]=%d\n",i, arr[i]);
    }
```

```
    int a = 3;
    int b = 0;
    int i = 0;
    int arr[10] = {9,4,3,3,1,8,4,2,5,9};
    for(i=0;i<=9;i=i+1){
        if (arr[i] == a){
            b = b + 1;
        }
    }
    printf("count␣of␣%d␣is␣%d\n", a, b);
```

以下，並び替えタスクの問題である．共通的な include 文と return 文は省略されている．また，コメントアウトの行は正解コードには含まれないダミーの行である．被験者は，それらの行を適切に処置（使用しないコードとして解答欄外に配置する）しなければならない．

```
int i = 0;
for(i=0;i<=6;i=i+2){
    printf("i␣is␣%d\n", i);
}
//  printf("end\n");
//  printf("a is %d\n", a);
//  for(i=0;i<=6;i=i+1){
//  for(i=0;i<=10;i=i+2){
//  if(i>2){
```

```
int i = 0;
for(i=0;i<=3;i=i+1){
    printf("aaa\n");
}
printf("end\n");
//  for(i=0;i<=2;i=i+2){
//  for(i=0;i<=2;i=i+1){
//  if(a > 3){
//  printf("%d\n", i);
//  printf("abc\n");
//  printf("a\n");
//  printf("a\n");
//  printf("a\n");
```

```
int a = 0;
int i = 0;
for(i=0;i<=4;i=i+1){
    a = a + i;
    printf("a␣is␣%d\n", a);
}
//  int a = 12;
//  if(a > i){
//  for(i=0;i<=3;i=i+1){
//  for(i=0;i<=10;i=i+2){
//  printf("i is %d\n", i);
//  printf("end\n");
//  a = 5;
//  a = i;
//  i = a + i;
```

```
int i = 0;
int j = 0;
for(i=0;i<=2;i=i+1){
    for(j=0;j<=2;j=j+1){
        printf("i=%d,␣j=%d\n", i, j);
    }
}
//  a = 10;
//  if(i == j){
//  for(i=0;i<=2;i=i+2){
//  for(j=0;j<=2;j=j+2){
//  printf("i is %d, j is %d\n", i, j);
//  printf("end\n");
//  a = i;
//  i = a + i;
```

以下，選択タスクの問題である．コード下部の 4 点のコメントアウトが選択肢である．一番目が正答である．

```
    int a = 10;
    int b = 20;
    printf("a␣is␣%d\nb␣is␣%d\n", a, b);
    printf("a␣+␣b␣=␣%d", a + b);
    printf("␣and␣");
    printf("a␣*␣b␣=␣%d\n", a * b);
    //-----
    /*
    a is 10
    b is 20
    a + b = 30 and a * b = 200
    */
    //-----
    /*
    a is 10
    b is 20
    a + b = 30
     and
    a * b = 200
    */
    //-----
    /*
    a is 10 b is 20
    a + b = 30 and a * b = 200
    */
    //-----
    /*
    a is 10
    b is 20
    a + b = 30
    a * b = 200
    */
```

```
    int a = 10;
    int b = 20;
    if (a > b){
        printf("A\n");
    } else {
        printf("B\n");
    }
    //-----
    /*
    B
    */
    //-----
    /*
    A
    */
    //-----
    /*
    A
    B
    */
    //-----
    /*
    AB
    */
```

```
    int a = 10;
    int b = 20;
    int c = 100;
    if (a + b > 20 && a * b > c) {
        printf("ABC\n");
    }

    if (c > a && c > b) {
```

```
    printf("DEF\n");
}

if (a == b || a * 10 == c) {
    printf("GHI\n");
}
//-----
/*
ABC
DEF
GHI
*/
//-----
/*
ABC
*/
//-----
/*
ABC
DEF
*/
//-----
/*
DEF
*/
```

```
int i = 0;
for(i=0;i<=3;i=i+1){
    printf("i␣is␣%d\n", i);
}
//-----
/*
i is 0
i is 1
i is 2
i is 3
*/
//-----
/*
i is 0
*/
//-----
/*
i is 3
*/
//-----
/*
i is 0
i is 1
i is 2
i is 3
i is 4
*/
```

```
int i = 0;
for(i=0;i<=2;i=i+1){
    printf("abc\n");
}
//-----
/*
abc
abc
abc
*/
//-----
/*
abc
```

```
*/
//-----
/*
abcabcabc
*/
//-----
/*
0
1
2
*/
```

```
int i = 0;
for(i=12;i<=15;i=i+1){
    printf("%d\n", i);
}
//-----
/*
12
13
14
15
*/
//-----
/*
12
*/
//-----
/*
27
*/
//-----
/*
12
15
*/
```

```
int i = 0;
for(i=0;i<=6;i=i+2){
    printf("i␣is␣%d\n", i);
}
//-----
/*
i is 0
i is 2
i is 4
i is 6
*/
//-----
/*
i is 0
*/
//-----
/*
i is 6
*/
//-----
/*
i is 0
i is 1
i is 2
i is 3
i is 4
i is 5
i is 6
*/
```

```
int i = 0;
int arr[5] = {5,6,2,8,1};
for(i=0;i<=4;i=i+1){
    printf("%d\n", arr[i]);
}
//-----
/*
5
6
2
8
1
*/
//-----
/*
arr[i]
arr[i]
arr[i]
arr[i]
arr[i]
*/
//-----
/*
1
2
5
6
8
*/
//-----
/*
0
1
2
3
4
*/
```

```
int a = 0;
int i = 0;
for(i=0;i<=4;i=i+1){
    a = a + i;
}
printf("%d\n", a);
//-----
/*
10
*/
//-----
/*
0
*/
//-----
/*
4
*/
//-----
/*
5
*/
```

```
int a = 0;
int i = 0;
int arr[5] = {1,4,3,5,0};
for(i=0;i<=4;i=i+1){
    if(arr[i] > a){
```

```
            a = arr[i];
        }
    }
    printf("%d\n", a);
    //-----
    /*
    5
    */
    //-----
    /*
    0,1,2,3,4
    */
    //-----
    /*
    0
    */
    //-----
    /*
    1,4,3,5,0
    */
```

```
    int a = 0;
    int i = 0;
    int arr[5] = {10,5,2,3,5};
    for(i=0;i<=4;i=i+1){
        a = a + arr[i];
    }
    printf("%d\n", a);
    //-----
    /*
    25
    */
    //-----
    /*
    10,5,2,3,5
    */
    //-----
    /*
    10
    */
    //-----
    /*
    2
    */
```

```
    int i = 0;
    int j = 0;
    for(i=0;i<=1;i=i+1){
        for(j=0;j<=2;j=j+1){
            printf("i␣is␣%d,␣j␣is␣%d\n", i, j);
        }
    }
    //-----
    /*
    i is 0, j is 0
    i is 0, j is 1
    i is 0, j is 2
    i is 1, j is 0
    i is 1, j is 1
    i is 1, j is 2
    */
    //-----
    /*
    i is 0, j is 0
    */
    //-----
    /*
```

```
i is 1, j is 2
*/
//-----
/*
i is 0, j is 0
i is 1, j is 0
i is 0, j is 1
i is 1, j is 1
i is 0, j is 2
i is 1, j is 2
*/
```

```
int a = 0;
int i = 0;
int arr[5] = {8,2,3,4,0};
for(i=0;i<=4;i=i+1){
    if(arr[i] > 1){
        a = arr[i];
    }
}
printf("%d\n", a);
//-----
/*
4
*/
//-----
/*
8,2,3,4,0
*/
//-----
/*
0
*/
//-----
/*
8
*/
```

# 実験６に使用した問題

以下，事前試験の問題である．Write 1 to 4 は出力結果と望まれる条件が被験者に提示される問題である．出題の出力結果は長い問題であった．そのため，回答であるコードを以下に記載する．望まれる条件とは，次のような内容である．Write1 では [ printf(”i=%d, j=%d, k=%d\n”, i, j, k); ] を一度のみ使用して良い．以下に記載したものは事前試験のみであるが，事後試験も基本的には同様の内容である．また，Write1 以外では，共通的な include 文や return 文は省略されている．

```
// Write1
#include<stdio.h>
int main () {
  int i, j, k, m;
  for (i = 0; i < 3; i+=1) {
    for (j = 0; j < 3; j+=1) {
      for (k = 0; k < 3; k+=1) {
        printf("i=%d,␣j=%d,␣k=%d\n", i, j, k);
      }
    }
  }
  return 0;
}
```

```
// Write2
  int i, j, k, a=0;
  for (i = 0; i < 3; i+=1) {
    for (j = 0; j < 3; j+=1) {
      for (k = 0; k < 2; k+=1) {
        a += 2;
        printf("i=%d,␣j=%d,␣k=%d,␣a=%d\n", i, j, k, a);
      }
    }
  }
```

```
// Write3
  int a, b, c, d;
  for (a = 5; a < 8; a+=1) {
    for (b = 10; b < 15; b+=2) {
      for (c = 5; c > 3; c-=1) {
        for (d = 0; d < 3; d+=1) {
          printf("a=%d,␣b=%d,␣c=%d,␣d=%d\n", a, b, c, d);
        }
      }
    }
  }
  printf("end\n");
```

```
// Write4
 int i, j, k, a = 0;
 for (i = 0; i < 3; i+=1) {
   for (j = 0; j < 3; j+=1) {
     for (k = 0; k < 3; k+=1) {
       a += i;
       printf("i=%d,␣j=%d,␣k=%d,␣a=%d\n", i, j, k, a);
     }
   }
 }
 printf("a=%d\n", a);
```

```
// Trace5
 int i, j, k;
 for (i = 0; i < 2; i+=1) {
   for (j = 0; j < 2; j+=1) {
     for (k = 0; k < 3; k+=1) {
       printf("i=%d,␣j=%d,␣k=%d\n", i, j, k);
     }
   }
 }
```

```
// Trace6
 int i, j, k, a=0;
 for (i = 0; i < 3; i+=1) {
   for (j = 0; j < 3; j+=1) {
     for (k = 0; k < 3; k+=1) {
       a += i;
     }
   }
 }
 printf("a=%d\n", a);
```

```
// Trace7
 int i, j, k;
 for (i = 5; i > 3; i-=1) {
   for (j = 10; j > 6; j-=2) {
     for (k = 100; k > 98; k-=1) {
       printf("i=%d,␣j=%d,␣k=%d\n", i, j, k);
     }
   }
 }
 printf("end\n");
```

```
// Trace8 pre-test
 int a,b;
 a = 10;
 b = 20;
 if (a > b) {
   printf("a=%d\n", a);
 }
 printf("end\n");
```

```
// Trace8 post-test
 int i, j, k, a=0;
 for (i = 0; i < 3; i+=1) {
   for (j = 2; j < 5; j+=1) {
     for (k = 0; k < 3; k+=1) {
       if (k == 2) {
         a += k;
       }
     }
   }
```

```
  }
 printf("a=%d\n", a);
```

```
// Trace9
 int i, j, k;
 for (i = 1; i < 3; i+=1) {
   if (i==1) {
     for (j = 0; j < 3; j+=1) {
       if (j==1) {
         for (k = 0; k < 2; k+=1) {
           printf("i=%d,␣j=%d,␣k=%d\n", i, j, k);
         }
       }
     }
   }
 }
```

以下は，反復群の学習に使用された一部の問題である．また，これらの問題は Visualize 群にも使用された．共通的な include 文や return 文は省略されている．

```
// Main code 1 (code 'a' at Table II)
 int i, j, k;
 for (i = 0; i < 2; i+=1) {
   for (j = 0; j < 3; j+=1) {
     for (k = 0; k < 3; k+=1) {
       printf("i=%d,␣j=%d,␣k=%d\n", i, j, k);
     }
   }
 }
```

```
// Arranged code of 'a' (code 'e' at Table II)
  int i, j, k;
 for (i = 10; i > 5; i-=2) {
   for (j = 0; j < 3; j+=1) {
     for (k = 0; k < 2; k+=1) {
       printf("i=%d,␣j=%d,␣k=%d\n", i, j, k);
     }
   }
 }
```

```
// Another arranged code of 'a' (code 'f' at Table II)
 int i, j, k;
 for (i = 0; i < 2; i+=1) {
   for (j = 0; j < 3; j+=1) {
     for (k = 0; k < 3; k+=1) {
       printf("i=%d,␣j=%d,␣k=%d\n", i, j, k);
       printf("abc\n");
     }
   }
 }
```

```
// Main code 2 (code 'A' at Table II)
 int i, j, k, a=0;
 for (i = 0; i < 2; i+=1) {
   for (j = 0; j < 3; j+=1) {
     for (k = 0; k < 3; k+=1) {
       a += 1;
       printf("a=%d\n", a);
      }
   }
 }
 printf("finish:a=%d\n", a);
```

```
// Arranged code of 'A' (code 'C' at Table II)
 int i, j, k, a=100;
 for (i = 0; i < 2; i+=1) {
   for (j = 0; j < 3; j+=1) {
     for (k = 0; k < 3; k+=1) {
       a -= k;
       printf("a=%d\n", a);
      }
   }
 }
 printf("finish:a=%d\n", a);
```

# 謝辞

本研究を行うにあたり，指導教員の濱本和彦教授には，本研究を始める段階から大変深い配慮で本研究を見守っていただき，また研究方針についてアドバイスをいただき大変感謝いたします．現時点の知識で振り返ると，当初の漠然とした方針しかないにもかかわらず本研究のサポートをいただきましたことを感謝いたします．特に研究の初期段階からアフォーダンスというキーワードについて本研究の根底に関わる点である旨につき，ご理解いただいたことは大変大きな励みになりました．

また，濱本研究室の方々には研究に対する意見を頂き，各種の実験に被験者としてご協力を頂きましたことを感謝申し上げます．皆様のご協力がなければ，本研究は何も進みませんでした．同様に，各種の実験に被験者としてご協力いただきました東海大学情報通信学部の方々にも厚く御礼申し上げます．

東海大学情報通信学部情報メディア学科 保坂憲一教授には，研究の初期段階から方針について相談に乗っていただき，また被験者募集の声かけにも都度ご協力いただきましたこと心より御礼申し上げます．その他にも，情報通信学部の各教授には，被験者募集の声かけにご協力いただきましたことをお礼申し上げます．

博士学位申請にあたり，学位審査委員をお引き受け頂きました，東海大学情報通信学部組込みソフトウェア工学科 渡辺晴美教授，東海大学情報通信学部通信ネットワーク工学科 石井啓之教授，東海大学情報通信学部通信ネットワーク工学科 高山佳久教授，東海大学情報通信学部情報メディア学科 伴野明教授，東海大学情報通信学部情報メディア学科 山田光穂教授 には，本研究を博士論文として纏める過程で，大変貴重な意見を頂きました．ご指摘頂きました点のお陰で，本研究の目的をより明確に定義することが出来，実験を通した本研究の改善点もより強調して説明が可能になりました．そして，動機づけという点についての改善も，ご指摘頂いた事で，提案手法の特徴として明確に強調する方針に気づかせていただきました．その他にも，基本的な事項ではありますが，博士論文のような長い論文をまとめるにあたり，十分な

配慮が欠けてしまった点である，各章のつなぎの意識や，各実験において仮説の強調と結果の明確化，提案手法の特徴の明確化，専門用語や実験の結果表の読み方関する説明不足の改善など，本論文を多くの方に理解していただくための貴重なご指摘を多数いただきました．先生方のご指摘により，本研究の特徴を明確に出来，よりわかりやすい説明を行えるようになりました事を，深く感謝申し上げます．

皆様のおかげで本研究を進めることが出来，最終的に形としてまとめることが出来ました．心より感謝申し上げます．