「主体的プログラミング学習段階」を対象としたプログラミング演習における学習困難性の考察

岡本 雅子 [1)]

1) 京都大学 高等教育研究開発推進センター

okamoto.masako.8v@kyoto-u.ac.jp

Analysis of Learning Difficulty for "Independently Learning Phase of Computer Programming"

Masako Okamoto[1)]

1) Center for the Promotion of Excellence in Higher Education, Kyoto University

概要

プログラミングの応用段階に向けた学習段階、すなわち自立的なプログラミング能力の涵養を目的とした学習段階において、演習形式のプログラミングの授業が提供できる学習内容について考察し、その内容を前提に構成した演習の実践例から、同段階の学習における困難性の具体的事例を抽出してその要因を考察する。

1 はじめに

プログラミング教育の必修化を前にして、プログラミング入門課程に係る研究が数多く試みられるようになった。こうした取り組みでは、基礎的な概念やコーディングを学ぶ際の問題点を学習者側からの視点で「つまずき箇所の類型」を調査したり [1]、教授者側の授業設計を課題として、その改善点やよりよい教授法などについて実践的に検証している [2]。ただし、例えば、小学校では、プログラミング的思考の習得を目的とするなど、これらの入門課程における授業はプログラミング能力の習得それ自体を直接の目的にしておらず、当然だが、この入門課程だけを学んだところで自力でプログラミングできるようになるわけではない。

「アルゴリズムや文法だけを教えるのであれば、個々のテーマごとに最適の例題や演習課題を用意するだけで、十分に対応することができる」「このような細切れの課題対応型に期待できるのは『コードの書き方』の修得までであろう」[3] というように、入門課程の授業で到達できる段階は凡そ、個々のサンプルを読み解く基礎が身につく程度である。

そして、入門課程の学習の後、自ら目的とするプログラムを設計するには、より実用的なプログラミング能力のトレーニングが必要となる。こうした「応用段階」のプログラミング教育については、授業あるいは演習をどのように実施したらよいかという授業設計のあり方を主な課題として議論されてきたが、一方、学習者側からの視点でどのような困難性がみられるかについて議論がなされてこなかった。

こうしたことから、本稿では、まず、模倣でなく自らプログラムを構想して記述するプログラミング演習において、学習者が具体的に作業する内容を階層化および明確化し、それらの作業のうち授業内においてどのようなトレーニングが実現できるかについて議論する。そして次に、この議論を踏まえて設計し、実施したプログラミング実践において、学習者にとってどのような困難性がみられたかについて抽出し、その問題点について考察することとしたい。

2 自立的なプログラミングに必要なプロセス

学習の目的が、自らが望むプログラムを作成できる能力を身につけるということであるならば、まず、プログラミング学習の段階について明確にし、プログラム作成のプロセスがどのようなものであるかについて明示してから議論を進めていく必要があると著者は考える。こうしたことから、本節では、最初に学習段階について述べ、次にプログラム作成のプロセスについて論じ、そのプロセスごとに必要とされる能力とそれを涵養する学習内容について、プログラミング演習において実現できる範囲がどこまでであるかといった視点から議論したい。

2.1 「依存的プログラミング学習段階」と「主体的プログラミング学習段階」

これまで、プログラミング教育に係る研究では、著者も含め、明確に定義しないままに、一般教育、専門教育、基礎、応用などの概念を用いてきた。例えば、専門教育という文言であれば、専門課程に在籍する学生が学ぶ内容という学習者の属性を意味するだけで、学習内容や学習段階そのものを示した概念ではない。その原因としては、研究者間においてこれらの概念が一定の幅をもって受け入れられていて、統一し、定義しようという試みがなされなかったからだと著者は考えるが、そうした結果、同じ文言を用いていても、対象とする学習内容がまちまちであったり、異なる文言を用いていても同じ学習内容を示す場合もあり、議論が成立しにくい状況であったことは否めない。こうしたことから、著者は、学習者の属性ではなく、学習の際の作業内容を軸に、学習段階を明確に定義してから議論を進めたいと思う。

初学者は、まず、最初にプログラミングの基本的な要素である変数と制御構造などを学習し、サンプルプログラムを模倣する「写経型学習」[4] 過程やサンプルプログラムの一部改変などに取り組む。これらは教員から与えられた知識を覚え、また、提示されたプログラムをタイピングして実行し、動作を確認する極めて依存的な学習過程である。この段階を経た学習者は、教授者から使用するコードや目的とするプログラムのイメージやモデルについて参考となる情報を与えられながらも、自らアルゴリズムを考え、それをコーディングするという主体的なプログラミング学習段階へと進む。本稿では、前者を「依存的プログラミング学習段階」、後者を「主体的プログラミング学習段階」と規定し、以降の論を進めたい。

2.2 コンピュータプログラムが完成するまでのプロセスと必要とされる能力

プログラムの作成は、まず、その目的を明確化することから始まる。例えば、企業であれば、クライアントからその目的を聞き出して（ヒアリング）、「要求定義」を明確にする。さらに、要求定義をさらに個別化した詳細仕様書を作成することで全体をモデル構築する。ヒアリングから始まるこの過程は、小規模な設計であれば、ひとりのプログラム作成者がその頭の中で進めている場合もあるわけだから、傍から見て顕在化していない作業の場合もあるだろうが、目的を明確化し、目的とする動作を総体として抽象化するというモデル構築の作業については、プログラムの規模の大小にかかわらず同様に必要とされるプロセスである。

続いて、このモデルをコンピュータの処理の形式に合わせてアルゴリズムを作成してからコーディングする。これは、個人が趣味でプログラムを作成する際には行わない作業ではあるが、目的を明確化する作業が必要であるという点については同様である。

最後に、実務レベルでは、必ず、仕様書にそってテストケースを作成して入念に動作を確認し、エラーがあった場合はプログラムを修正する。

このように考えると、プログラムの作成は、大きく分けて次の 4 つの段階があると表現できる。

- 第一に、具体的事象の抽象化の過程（モデル構築）
- 第二に、モデルをコンピュータ処理に適合させるアルゴリズム化の過程（アルゴリズム化）
- 第三に、アルゴリズムをプログラミング言語に変換するコーディングの過程（コーディング）
- 第四に、動作テストと修正の過程（テスト）

2.2.1 モデル構築

モデル構築については、具体的事象を抽象化し、論理化する能力が必要とされる。これらの能力は、コンピュータプログラムの作成にかかわらず、学問領域において広範に必要とされ、そしてそれらの学習を介しても涵養される汎用性の高い能力である。そのため、これらの能力の涵養について、プログラミング学習が単独で担わなければならない、あるいは、単独で担うことができるような学習範囲であるとまでは言えないが、演習内での作業がその能力の涵養に寄与することまでは否定できないことから、プログラミング演習内で試みる価値がある学習内容であると著者は考える。

2.2.2 アルゴリズム化

アルゴリズムとは、「問題を解くための手順や特定の処理の手順を明確に定義したもの」であり、コンピュータが処理する一連の流れを示したものである。コンピュータに意図する処理を正しく実行させるために適切な指示が必要なため、この適切な指示を定式化させたものがアルゴリズムであるが、プロセスとして見るならば、アルゴリズム構築は、構築したモデルをコンピュータ特有の処理形態に合わせる作業とも言える。なお、モデル構築とアルゴリズム化を弁別せずに同一の過程として取り扱い、「アルゴリズムの作成」と見做すケースも散見されるが、本稿では、これらの作業に必要とされる能力が異なることから、それぞれを分けて個別に扱っている。

アルゴリズム化の作業は、基礎的なあるいは根源的

な思考力を必要とするというよりは、コンピュータ特有の処理形態について理解してさえいれば、構築したモデルに当てはめるだけで対応が可能である場合も少なくない。

こうしたことから、アルゴリズムは、プログラミング学習の中で学ばなければ他に機会を得られない学習範囲であり、また、プログラミングに係る授業の中でこそ学習が可能なプロセスであると著者は考える。

2.2.3 コーディング

依存的プログラミング学習段階の演習では、学習者がサンプルプログラムを模倣して入力、実行し、結果を確認することを重視する場合が多いが、和田 [3] が指摘しているように、実践的なプログラミング能力を身に付けるにはこうした学習だけでは十分でないとされている。入門課程を経た学習者は、数十行から 100 行程度のまとまったプログラムを作成することにより、複雑な処理の記述方法を学習していくわけであるが、コーディングについて十分な知識を獲得するには、単にプログラムを書かせるだけでなく、その論理的な理解ができているかをこまめにチェックしながら、プログラム全体を俯瞰して確認するという作業も有効であり、こうした内容は、授業としての演習内で十分に実現可能であると著者は考える。

2.2.4 テスト

テストケースを適切に設計してプログラムの誤りを発見することや仕様を満たしているかを確認するなど、テスト技法を学ぶことも重要ではある。ここでは、アルゴリズムに沿ってコーディングを終えた段階で想定していた動作を確認するだけではなく、様々な入力内容を試すことで想定しなかった動作の有無も確認することになる。従来の初学者を対象とした導入教育のようにプログラムの入力と出力が記述された実行例が学習者に提示するケースでは、実行例と同じ入出力だけを確認する場合がほとんどであるが、テストをする能力の涵養を目的とするならば、まず、入力のケースを想定することから学ぶ必要があるし、それ自体が本来的な学習の対象であるともいえる。ただし、授業時間内で作成できるような短いプログラムを作成した場合ならば、想定される入力のケースの範囲は限定されており、コーディングを終えた段階で想定していた動作を確認するという初学者が学ぶ内容とほとんど変わりない作業を実施するだけになってしまう。もちろん、デバッグなどについて慣れていくこと、経験を積むことは演習を介しても十分に可能であるし、こうした部分についてはプログラミングの授業、あるいはプログラミング演習が担う範囲であるが、テストを全体的に学習しようと考えるならば、数千行におよぶような大規模なプログラムを作成することが条件となるため、とくに長期にわたって実施することを前提とした教育課程や OJT がその場として適当であると著者は考える。

表 1　対象授業の履修者数

	木 2	金 1	金 2	合計
2018 年後期	75	60	61	196

3　授業としてのプログラミング演習で実現可能な学習の範囲

前節では、プログラミング学習の段階を「依存的プログラミング学習段階」と「主体的プログラミング学習段階」に分けるとともに、プログラムが完成するまでの作業を流れに沿って類型化し、とくに大学などで授業として実施されるプログラミング演習において、学習者が自立的なプログラミング能力を修得することを目的とした場合にはどのような学習内容が提供できるか、あるいはどのような学習内容を提供すべきかといった観点から検討した。

こうした結果、著者は、

- 具体的事象を提示した状況下におけるモデルの構築
- モデルに沿ったアルゴリズムの作成
- 使用する複数のコードを予め提示した上で実施するコーディング
- デバッグを主な目的としたテスト

の 4 点については、「主体的プログラミング学習段階」における授業あるいは演習において提供可能であり、提供すべき内容であると考え、それらの能力向上を目的とした授業設計を試みることとした。

4　授業実践

4.1　対象とした授業

本研究の対象とした授業は、2018 年度の後期に京都大学で実施された全学共通科目（農学部が独自に提供するクラス指定の科目で 1 回生から 4 回生まで受講可）「情報基礎演習」である。表 1 に示す 3 クラスを対象とした。

本科目は、教養教育におけるコンピュータリテラシ教育という位置付けで週 1 回 90 分の選択科目として

実施したもので、「アカデミックな活動に必要な ICT スキルを獲得すること」、「自立した ICT ユーザとなること」、「ICT スキルを自主的・継続的に獲得する自学自習能力を身につけること」の 3 点を目的としている。コンピュータやネットワークといった教育・学習における ICT 利用に関する基礎知識を獲得するとともに、学術情報の探索、学術文書の作成、表計算ソフトを用いた分析やシミュレーション、プレゼンテーション、プログラミングについては演習形式で実施している。プログラミングの授業は、後半 4 回の授業を割り当てた。このような科目構成からプログラミングのみを学ぶ授業に比べ、相対的に時間的制約は大きい状況であった。

4.2 授業デザインと授業形態

対象とした科目の中でプログラミングを学ぶ機会は、先述のように 1 授業 90 分で計 4 回しかなく、学習する時間が限られていることから、学習内容は、前半 3 回の授業でプログラミングの基本的な要素である変数と制御構造などを学習し、最終の 4 回目である程度まとまったプログラムを作成した（表 2 参照）。1 回目から 4 回目までを「依存的プログラミング学習段階」に充て、4 回目を「主体的プログラミング学習段階」に充てる構成である。なお、本研究では、受講生自らプログラムを作成する第 4 回目の授業を分析対象とした。

また、本授業では、講義とテキスト教材による個別学習を取り入れた授業形態を採用し、講義では短時間にポイントを絞って解説を行った。受講生がつまずいた場合は、教員やティーチングアシスタント（1 人）がサポートに入り、直接対応した。なお、演習には、京都大学の教育用コンピュータ端末を用い、Python の環境には Web ブラウザ上にエディタと Python の実行環境が搭載されている「Jupyter Notebook」を用いた。

本研究の対象とした第 4 回目の授業は、「数当てゲームの作成」を題材とした。90 分の授業構成は以下の通りである。

(1) 数当てゲームの exe ファイルを配布し、各自が動作を確認する
　→提示された具体的事象をモデル化
(2) 作確認後、アルゴリズム（プログラムの動く手順）を考えて、アルゴリズムは、フローチャートの図や文章、矢印などで可視化する
　→アルゴリズム化
(3)（モデル構築に失敗した受講生らを対象に）モデルを提示し、それを元にしたアルゴリズム化
(4) Python のプログラムを作成する際に必要となるコードなどの要素の確認
(5)（アルゴリズム化に失敗した受講生らを対象に）ランダムな値の生成方法や入力された値のチェックなど数当てゲームのアルゴリズムの確認
(6) プログラムの作成
(7) プログラムの動作確認と修正

表 2　プログラミングの授業内容

	授業内容
1 回目	• 開発環境の基本操作 • Python の基礎
2 回目	• 条件を判断して処理を実行する • 例外処理など
3 回目	• 同じ処理を指定した回数だけ繰り返す • 条件が満たされている間に処理を繰り返すなど
4 回目	• 数当てゲームの作成

(2) の「アルゴリズム」に関しては、授業の開始時に A4 の用紙を配布した。受講生には手書きでアルゴリズムを記入するように指示し、授業の最後にその用紙を回収した。なお、受講生には、(3) の解説後に追記しないようにアナウンスしている。

また、第 4 回目の「数当てゲーム」は、4 桁の数字を当てるゲームで、2 人のプレーヤーが「出題者」と「回答者」にわかれてプレイするものである。本実践で採用した「数当てゲーム」のルールは以下の通りである。

1. 出題者は、「0」から「9」の数字を使って、4 桁の数字を考える（同じ数字を 2 個以上使用可能）
2. 回答者はその数字を予想して、出題者に提示する
3. 出題者は提示された数字を判定し、ヒットとブローの数を回答者に提示する
4. 2〜3 を繰り返し、（ヒットとブローの結果を参考に）回答者は、出題者が考えた 4 桁の数字を当てる

4.3 受講生の構成

本実践での対象とした受講生は、プログラミングの第 4 回目の授業に出席し、課題を提出した 133 人である（表 3 参照）。なお、この授業はプログラミング初学者を想定したものであるが、経験者の履修も可能であり、本実践では、Java など他のプログラミング言語の

表 3　本実践の対象者

	木 2	金 1	金 2	合計
2018 年後期	54 (10)	39 (3)	40 (8)	133 (21)

※括弧内はプログラミング経験者

表 4　アルゴリズムの学習状況

	木 2	金 1	金 2	合計
100 点	30 (9)	21 (3)	22 (3)	73 (15)
100 点未満	24 (1)	18 (0)	18 (4)	60 (5)

※括弧内はプログラミング経験者

既習者が 21 人であった。

4.4　検証方法

自立的なプログラミング段階の学習における困難性の具体的事例を抽出するため、第 4 回目の授業で実施した数当てゲームについて、受講生から提出された「アルゴリズム」と「プログラム」に関し、つまずき箇所とその件数、内容について調べた。さらに、プログラミング学習に対する受講生の率直な意見を集めるために、第 4 回目の授業終了後に自由記述アンケート（授業の感想）を実施した。

5　結果と考察

5.1　モデル構築とアルゴリズム化について

本授業では、第 4 回目の数当てゲームのアルゴリズムについて 133 人から提出があった（表 4 参照）。このうち、プログラミング経験者 15 人を含む 73 人は正しく記述できていた。一方、残りの 60 人は、表 5 に示すように「白紙で提出している」「繰り返し処理が抜けている」「繰り返し処理の位置に誤りがある」「一部の処理が抜けている」などのつまずきが見られた。

具体的には、アルゴリズムの用紙に「開始」と「終了」のみ記載して、中身の処理が全く記載されていないものが 16 人いた。次に、「4 桁の乱数を発生させる」「ヒットの場合は加算する」などの処理のみパーツとして表現しているものが 18 人いた。この 18 名に関しては、数当てゲームをモデル化して表現することができなかったと推察される。また、数当てゲームのモデル化ができ、全体の処理を可視化できているが、繰り返し処理が抜けているものが 38 名であった。これは、モデル化した数当てゲームを適切にアルゴリズムに落とすことができていないものと考えられる。本実践では、モデル構築とアルゴリズム化を同時に課題としたが、モデル構築の時点でつまずいている学習者とモデル構築はできてもアルゴリズム化に失敗したものとに分かれ、双方は別個のつまずき箇所として表出している。

5.2　コーディングについて

次に、モデル化に失敗したためにアルゴリズム化に進めなかった受講生に対する救済処置として、「数あてゲーム」のゲームモデルを説明し、アルゴリズムを解説した。これを踏まえてコーディングの段階に進めている。

数当てゲームに関するプログラムはアルゴリズムと同様に 133 人から提出された。表 6 に示すように、正しく動作するものを提出できた受講生は 86 人であった。残りの 47 人は、文法エラーや変数の初期化がされていない、繰り返し処理が入っていないなどの誤りが見られた（表 9 参照）。具体的な誤りは表 9 に示す通り、「変数名の重複」が 2 人、「変数の初期化がされていない」が 11 人、「文法エラー」が 8 人、「繰り返し処理がない」が 25 人、「各パーツのプログラムのみ作成（統合されていない）」した人が 2 人、「ファイルが壊れていた」が 8 人であった。

次に、アルゴリズムの得点が 100 点未満であった受講生 60 人のプログラムの得点を表 7 に示す。60 人のうち 30 人が 100 点であったが、残りの 30 人（プログラミング経験者 1 人を含む）が 100 点未満であった。

アルゴリズム化でつまずいた受講生は、数当てゲームのアルゴリズムの解説を聞くことにより、30 人がプログラムを完成させた（表 7 参照）。一方、アルゴリズム化でつまずいた受講生のうち、30 人がプログラムを完成させることができなかった。

アルゴリズムの得点が 100 点未満で、さらにプログラムの得点が 100 点未満であった受講生のつまずきの事例を表 11 に示す。

加えて、アルゴリズム化でつまずいた内容とプログラムの点数を表 10 に示す。アルゴリズムを「白紙」で提出し、プログラムの点数が 100 点未満であった 10 人のうち、「繰り返し処理がない」ものが 8 名、「ファイルが壊れていた」ものが 2 名であった。「一部の処理のみ記載していた」6 名のうち、「文法エラー」が 3 名、「繰り返し処理がない」ものが 2 名、「ファイルが壊れていた」ものが 1 名であった。次に、「全体の処理に誤りはないが、ゲームを続けるための繰り返し処理が抜けていた」14 名のうち、「変数名の重複」が 2 名、「変数の初期化がされていない」が 4 名、「文法エラー」が 4 人、「繰り返し処理がない」ものが 3 人、「ファイ

表 5　アルゴリズム化でのつまずき状況

つまずきの事例	つまずきの件数			
	木 2	金 1	金 2	合計
白紙で提出した	1	10	5	16
一部の処理のみ記載していた	7	4	7	18
全体の処理に誤りはないが、ゲームを続けるための繰り返し処理が抜けていた	17 (1)	11	10 (4)	38 (5)

※括弧内はプログラミング経験者

表 6　プログラムの学習状況

	木 2	金 1	金 2	合計
100 点	35 (10)	21 (2)	30 (5)	86 (17)
100 点未満	19 (0)	18 (1)	10 (2)	47 (3)

※括弧内はプログラミング経験者

表 7　アルゴリズム 100 点未満の受講生のプログラムの点数

	木 2	金 1	金 2	合計
100 点	10 (1)	7 (0)	13 (3)	30 (4)
100 点未満	14 (0)	11 (0)	5 (1)	30 (1)

※括弧内はプログラミング経験者

表 8　アルゴリズム 100 点の受講生のプログラムの点数

	木 2	金 1	金 2	合計
100 点	25 (9)	15 (2)	17 (2)	57 (13)
100 点未満	5 (0)	6 (1)	5 (1)	16 (2)

※括弧内はプログラミング経験者

ルが壊れていた」ものが 1 名であった。

とりわけ、「繰り返し処理」の不備に関してはプログラミング経験者 2 名も含まれており、加えて、自由記述アンケートに「繰り返し処理が難しく感じた」などの意見が見られるなど、繰り返し処理を実際に活用することは難易度が高かったことが推察される。

さらに、アルゴリズムが 100 点であった受講生 73 人のプログラムの得点を表 8 に示す。73 人のうち 57 人がプログラムの作成に関しても 100 点であったが、16 人（プログラミング経験者 2 人を含む）は 100 点未満であった。具体的な誤りは表 12 に示す通り、「変数の初期化がされていない」が 4 人、「文法エラー」が 2 人、「繰り返し処理がない」が 8 人、「各パーツのプログラムのみ作成（統合されていない）」した人が 1 人、「ファイルが壊れていた」人が 3 人であった。

「変数の初期化がされていない」については、変数を初期化することを理解していないことが考えられるため、プログラミングの知識が不足していたと推察される。次に、「文法エラー」については、デバッグのスキルが不足していたために、文法の誤りを見つけられなかった、あるいは誤りを見つけてもプログラミングの知識が不足しているために修正することができなかったと考えられる。また、「繰り返し処理がない」ものが 8 名もいた。この 8 名は、アルゴリズム化する際は、繰り返し処理を記述できていたにも関わらず、実際のプログラムで記述することができなかった。これについては、アルゴリズムをコーディングする過程につまずく要因があるかコードに関する理解が不足しているかのどちらかであると考えられるが、本実践の結果からは判別する手がかりが得られなかった。加えて、各パーツのプログラムのみ作成（統合されていない）」ものが 1 名おり、同様にアルゴリズム化はできているにも関わらず、コーディングできていないことから、繰り返し処理と同様に 2 種類の可能性が考えられるが、どちらの要素に起因するかについては手がかりがないため判別できない。

6　まとめと今後の課題

本研究では、教授者の補助を得ながらであっても自らプログラムを作成する学習段階を「主体的プログラミング学習段階」と規定し、その段階の学習内容をモデル構築、アルゴリズム化、コーディング、テストと 4 段階に階層化して、それぞれの段階におけるつまずき箇所を抽出した。その結果、自らプログラムを構想して記述していくことができないという事象であっても、その要因はまちまちであり、各学習階層において別個の困難性として表出していることがわかった。とくにアルゴリズムをチャートとして記述できているにもかかわらずコーディングの際に抜け落ちていた学習

表 9　コーディングでのつまずき状況

つまずきの事例	つまずきの件数（重複あり）			
	木 2	金 1	金 2	合計
変数名の重複	2	0	0	2
変数の初期化がされていない	4	4	3	11
文法エラー	5	3	0	8
繰り返し処理がない	10	10	5 (2)	25 (2)
各パーツのプログラムのみ作成されている（統合されていない）	2	0	0	2
ファイルが壊れていた	2	3 (1)	3	8 (1)

※括弧内はプログラミング経験者

表 10　アルゴリズム化でのつまずき状況とプログラムの点数

つまずきの事例	プログラムの点数		
	100 点	100 点未満	合計
白紙で提出した	6	10	16
一部の処理のみ記載していた	12	6	18
全体の処理に誤りはないが、ゲームを続けるための繰り返し処理が抜けていた	12 (2)	14 (3)	26 (5)

※括弧内はプログラミング経験者

表 11　アルゴリズム 100 点未満の受講生のコーディングでのつまずき状況

つまずきの事例	つまずきの件数（重複あり）			
	木 2	金 1	金 2	合計
変数名の重複	2	0	0	2
変数の初期化がされていない	4	3	0	7
文法エラー	4	2	0	6
繰り返し処理がない	6	7	4 (1)	17 (1)
各パーツのプログラムのみ作成されている（統合されていない）	1	0	0	1
ファイルが壊れていた	1	2	2	5

※括弧内はプログラミング経験者

者が見られたことは興味深い。こうしたコーディング段階の誤りの原因が、単に当該コードの理解不足であるならば、当該コードを知識として、再度、記憶すればよいのであるが、アルゴリズムをもとにをコーディングする際に本実践において明らかにできなかった想定外の要因があるとも考えられ、これについては今後の研究課題としたい。

自らプログラムを構想し、それをプログラミング言語に落としていく作業は、上流から下流へとつながる一連の作業であるため、上流のどこでつまずいたとしても、それに続く下流の作業に進むことはできない。上流から下流までのそれぞれの作業は、どれをとっても自立的にプログラミングする際には必要とされる能力であるが、その能力を涵養しようとする授業あるいは演習において、その一つが欠けるからといって、以降の学習に手を付けられないようでは学習者の不利益が大きいし、授業として成立しない。本実践でも明らかになったように、上流から下流までの各階層につまずき箇所があるのだから、続く階層に移行する際には、前階層の成功が次階層の成功の前提にならないように、つまり、どの階層からでも作業が可能であるように適切にフォローしながら授業を進めていく必要があるものと著者は考える。

表 12　アルゴリズム 100 点の受講生のコーディングでのつまずき状況

つまずきの事例	つまずきの件数（重複あり）			
	木 2	金 1	金 2	合計
変数名の重複	0	0	0	0
変数の初期化がされていない	0	1	3	4
文法エラー	1	1	3	4
繰り返し処理がない	4	3	1 (1)	8 (1)
各パーツのプログラムのみ作成されている（統合されていない）	1	0	0	1
ファイルが壊れていた	1	1 (1)	1	3 (1)

※括弧内はプログラミング経験者

謝辞

研究のためのフィールドをご提供いただいた京都大学高等教育研究開発推進センター酒井博之准教授に深く感謝申し上げます。

また、本研究は、科研費・若手研究 (B)（研究代表者：岡本雅子)「プログラミング演習の学習科学的分析と初学者向け学習教材の開発（17K17824)」の助成を受けたものである。

参考文献

[1] 岡本 雅子・喜多 一、プログラミングの「写経型学習」における初学者のつまずきの類型化とその考察、パイデイア、滋賀大学教育学部附属教育実践総合センター紀要、22、49-53、2014.

[2] 岡本雅子・村上正行・吉川直人・喜多一、「視覚的顕在化」に着目したプログラミング学習教材の開発と評価、日本教育工学会論文誌、37、1、35–45、2013.

[3] 和田 浩、スパイラル・アプローチによる C 言語プログラミング教育の実践と評価－受講生中心の学習を目指した試み、UNISYS TECHNOLOGY REVIEW、19、2、322-337、1999.

[4] 岡本雅子・村上正行・吉川直人・喜多一、プログラミングの写経型学習過程を対象としたつまずきの分析とテキスト教材の改善－作業の自立的遂行と作業を介した理解のための支援と工夫－、京都大学高等教育研究、19、47–57、2013.