Computational Thinking と小中学校での授業実践

古田貴久・町田真彦・高橋一成
荒川大地・奥木芳明

Computational thinking and its educational practices in elementary and junior-high schools

Takahisa FURUTA, Masahiko MACHIDA, Issei TAKAHASHI,
Daich ARAKAWA and Yoshiaki OKUGI

群馬大学教育学部紀要　芸術・技術・体育・生活科学編
第 55 巻　69―77 頁　2020　別刷

Computational Thinking と小中学校での授業実践

古　田　貴　久[1)]・町　田　真　彦[1)]・高　橋　一　成[1)]
荒　川　大　地[1)]・奥　木　芳　明[2)]
1）群馬大学教育学部
2）長野原町立北軽井沢小学校
（2019 年 9 月 25 日受理）

Computational thinking and its educational practices in elementary and junior-high schools

Takahisa FURUTA[1)], Masahiko MACHIDA[1)], Issei TAKAHASHI[1)],
Daich ARAKAWA[1)]and Yoshiaki OKUGI[2)]
1）Department of Technical Education, Faculty of Education, Gunma University
2）Kitakaruizawa Elementary-School, Naganohara
（Accepted on September 25th, 2019）

１．はじめに

2020 年度から本格実施される小学校の指導要領では、「プログラミング的思考」「アルゴリズム的思考」の能力の育成を目的として、コンピュータのプログラミングが授業に取り入れられる。「プログラミング的思考」とは、「自分が意図する一連の活動を実現するために、どのような動きの組合せが必要であり、一つ一つの動きに対応した記号を、どのように組み合わせたらいいのか、記号の組合せをどのように改善していけば、より意図した活動に近づくのか、といったことを論理的に考えていく力」（文部科学省，2017）とされるが、これは欧米で computational thinking（CT）と呼ばれる概念に対応する（赤堀，2018）。

CT は、Wing（2006）が最初に提唱した、コンピュータ科学の概念や用語に依拠して規定された問題解決の技能（skill）・能力（ability）である。問題解決の過程（process）を含めることもある。

CT には多くの研究者によっていくつもの定義が提案されており、その内容は提案者の専門分野によって、2 つの大きな系統がある。Computer Science（CS：コンピュータ科学）系の研究者は、CT について、コンピュータ科学とマッチした、日本の「プログラミング的思考」に近い定義を提案する。一方で、教育系の研究者と実務家にとって、そもそも教育とは人間を育てること（to raise a good citizen）であるから、より一般的な問題解決能力、論理的思考力の育成を念頭に置いた定義を提案している（Voogt, Fisser, Good, Mishra, & Yadav, 2015）。

本論文では、学校教育の現場を念頭におき、論理的思考力、問題解決能力の育成に沿った方向で CT とプログラミング的思考を考える。

２．Computational Thinking 登場の背景

CT と類似の概念は Wing（2006）以前から提唱されていた。それらに共通する基本的な発想は、論

理的で合理的な思考過程を定式化して教育に供するというものだと言えよう。

Polya（1957）は、身近な日常の問題も含めて、問題は処理可能なサイズに小さく分解するなどの4段階からなる「問題の解き方」を示し、数学に限らず幅広い問題を解決することに有効であると主張した。

構成主義的発想からPapert（1980；1993）は、プログラミング言語LOGOを開発し、幾何学概念の新しい学習法を示した。LOGOを使い、子どもは「タートル」と呼ばれるロボットに前進・後退などの指示を出す。タートルは移動しながら軌跡を残して円や三角形を描く。このような活動を通して、従来の算数の授業とは異なった視点に立った学習と理解が期待された。なお、今日、各国の小学校で盛んに利用されているScratchはLOGOの発展版である（Resnick, Maloney, Monroy-Hernandez, Rusk, Eastmond, Brennan, Millner, Rosenbaum, Silver, Silverman, & Kafai, 2009）。

diSessa（2000）は認知心理学的視点からLOGOプログラミングを通じて論理的思考能力や問題解決能力の育成を図れると論じ、そのような論理的思考能力や問題解決能力をcomputational literacyと呼んだ。

Computational ThinkingはもともとCSを学校教育に普及させ、大学のCS系学部への進学者を増やすためのものであった。

日本でも理科離れが言われるようになった1990年代後半から、各国で、大学のCS系学部への進学者の落ち込みが激しくなった。このため、将来のIT分野での競争力が低下することへの危機感から、子どもたちに将来の進学先・就職先としてIT・CS分野を選択肢に入れてもらうことが課題となった。

将来の職業選択にはmiddle school（中学校）段階が特に大きな影響があるとされる。そこで、中学生、とりわけこの分野は自分の分野ではないと考えがちな女子およびマイノリティの子どもたちを念頭に、彼らに学校の授業でコンピュータ科学というものを認知してもらい、興味・関心持ってもらおうという狙いのもと、CSの重要性を中学校に限らずK-12（幼稚園から高校まで）の先生たちにアピールし、実際に授業に取り入れてもらうため、カリキュラムを提案したり、子どもたちが楽しんでCSの基本的な概念に触れ、その重要性に気付いてもらう啓発活動が盛んになった。CTの提唱や、子どもにも使いやすいプログラミング言語の開発やゲームプログラミングの授業の提案、CS-unplugged（アンプラグド）というコンピュータを使わないCSなどは、そのような状況のもとで生まれたものである（Barr, Stephenson, 2011; Repenning, Ioannidou, 2008; Rodger, Hayes, Lezin, Qin, Nelson, Tucker, 2009; Royal Society 2012; Werner, Denner, Campe, 2012）。

なお、アンプラグドに対しては、ゲーム要素があって子どもが興味を持って取り組める、専門家が作った教材なので信頼できる、情報処理やコンピュータの基礎に触れるのに適当な題材なので、自分も授業で実施してみたいという声を聞くことがある。だが、授業者自身がCSをあまり理解していないのにアンプラグドを実施することは適当ではない。アンプラグドは、授業者の不勉強を補ってくれる魔法のような教材ではない。

3．教育研究者・実務家にとってのCT

プログラミング的思考やCTの大枠は前述の通りであるが、実践で必要になるより具体的で操作的な定義は、学校教育の研究者、実務家などの間でも一致を見ていない（Voogt, Fisser, Good, Mishra, & Yadav, 2015; Yadav, Stephenson, Hong, 2017）。CTを認知的側面に限定せず、21世紀を生きる子どもたちに身に着けて欲しい能力と広く定義し、論理的思考力や問題解決能力に加えて、創造性や他人との協調性を含めることもある（Doleck, Bazelais, Lemay, Saxena, Basnet, 2017; Korkmaz, Cakir, Yasar Ozden, 2017; Lu, & Fletcher, 2009; Mishra, Yadav, & the Deep-Play Research Group, 2013）。しかし、CTやプログラミング的思考が、問題解決活動であり論理的思考力や批判的思考力を含む点は共通している。

プログラミング的思考、あるいはCTを授業でいかに実践するかについて、小学校での本格実施を控

えて、多くの論考や実践例の提案がなされている（ほんの一例だが、例えば、赤堀，2018；文部科学省，2018；尾崎・伊藤，2017）。小学校のプログラミングでは Scratch が有力な選択肢であるが、NHK for school では、Scratch プログラミングの 10 分番組（Why!? プログラミング）を、一部は小学校・音楽、算数、理科、中学校・技術などに対応させて、指導用資料やワークシートや授業用資料とともに提供している（www.nhk.or.jp/sougou/programming/）。なお、NHK for school は、それ以外にも教育番組や映像資料の宝庫である。

CT やプログラミング的思考は、問題解決過程や論理的思考力と密接に関連している。ただし、CT やプログラミング的思考を学ぶことで問題解決過程や論理的思考力が獲得されるとすれば、それは原因と結果の関係が逆である。CT やプログラミング的思考は、人間が本来的に持っている問題解決過程や論理的思考力を、コンピュータ科学の用語や概念を用いて整理し定式化したものであって、CT やプログラミング思考を学んだから論理的に考えられるようになる、というわけではない。小学校低学年の児童でも、極めて高度な論理的思考能力や問題解決能力を発揮することは、トレーディングカードゲームやビデオゲーム、恐竜や宇宙に熱中する子どもたちがよい例である。CT やプログラミング的思考は、元来、状況次第、対象次第で高い論理的思考能力や問題解決能力を発揮する子どもたちに、彼らが無意識的に実行してきた思考のプロセスを意識させ、算数や国語などとは違った題材で明晰な言語使用や論理性に導き、思考を育てようとするものだと言えよう（diSessa, 2000）。このことは、中学校、高校と、学校段階が進んだとき、我々を取り巻く自然の世界、ならびに人工物や社会制度などから成る人為的な世界を、モデルを通して、システムとして理解することの基礎となろうという期待や信念がある。

4．Computational Thinking の技能的能力的構造

CT は広い意味では論理的合理的な問題解決の技能・能力、過程であり、プログラミング的思考という意味では、記号を組み合わせて一連の活動を構成することである。このような枠組みの中で、実際の授業を設計するためには、CT に加えて問題解決の方法論をもう少し具体的に理解する必要があろう。例えば、解決しようとする問題を直接的に解決しようとするのではなく、より小さなサイズの問題に分割して、1 つずつ解決していくことで、元々の問題の解決を図る、ということは、問題解決の基本である。問題の分割や分解は、CT の基本技能の 1 つである。

本論文では、小中学校での利用を念頭に置いた、標準 CT 評価尺度の作成経験をもとに、CT を構成する基本的技能・能力を「アルゴリズム」、「抽象化」、「評価」の 3 つとする。この構造は、CT の key process を「抽象化」と「プログラミング」の 2 つとした Computing at School Working Group（2012）に「評価」を追加した格好になっている。以下では、「アルゴリズム」と「抽象化」について、その内容と、これらが児童・生徒のどのような活動として現れるか説明する。なお、以下では、抽象化、アルゴリズム、分解、一般化などについて述べるが、これらは互いにきれいに分離できる概念ではない。抽象化は分解を含むし、アルゴリズム作りで抽象化は行われる。これらの技能・能力は重なり合っているのである。

4.1. アルゴリズム（algorithm）

アルゴリズムは、コンピュータ科学的には、特定の計算を正しく遂行することが保証された一連のステップである。各ステップは意味が完璧に規定された基本的な操作として表されていて、それが何を意味するかに曖昧さはない。さらに、必ず停止するという条件もつく（Kernighan, 2011）。ただし、アルゴリズムは元来コンピュータやプログラミングを前提としない。例えば、2 つの自然数の最大公約数を求めるアルゴリズムである「ユークリッドの互除法」は紀元前 300 年ごろの発明と考えられており、コンピュータが発明されるはるか前からアルゴリズムは存在したのである（増原，2006）。

しかしながら、アルゴリズムの定義と日本の「プログラミング的思考」は軌を一にするところから、プログラミング的思考はその名の通り、プログラミングをモデルにした思考様式であると言える。ただし、文部科学省が強調するように（文部科学省, 2018)、学校の授業にプログラミングを導入する狙いは、プログラムコードが書けるようになることではないとすれば、「プログラミング的思考」は教育系の CT の定義に沿って、論理的合理的な問題解決の方法論であると考えるべきであろう。その上で、アルゴリズムも、問題や課題を確実に解決する手順を、曖昧さを排して、すなわち、こういうときはどうしたらよいのか？と疑問を持たれたり、違う意味にとられる余地なく記述したもの、いわゆる段取りのことであると捉えるのが妥当だと考えられる。

CT におけるアルゴリズムは、まずは、問題解決過程のステップを、最初に何をして、次に何をして、さらにその次に何をするというふうに、正しい順番に並べることである。これは情報の用語で「順次処理」という。次に、状況や対象への柔軟性を持たせるために、もしこうであったらこうする、そうでなければこうするといった「条件判断」、「条件分岐」を含む。さらに、抽象化や一般化とも共通するが、問題解決過程の一部に同じ処理や作業を繰り返すことが含まれれば、「繰り返し処理」として明示することでもある。なお、「順次」、「条件分岐」などの用語は、中学校・技術で教える用語ではあるが、小学校段階では知らなくても支障はないであろう。

しばしば、「条件判断」や「条件分岐」は子どもに教えるには難しいのではないかという意見を聞く。だが、条件判断や条件分岐は、我々が普段、頻繁に行っていることである。例えば、「部屋が暗かったら電気をつける」というのは、「部屋は暗いか？」という条件判断である。もし条件が成立する（部屋が暗い）なら電気をつけるし、成立しなかったら（まだ明るい）電気をつけないというふうに分岐して、すなわち、違う処理を行っている。あるいは、「鍋のお湯が沸いたらスパゲッティを入れる」というのは、「火にかけた鍋の水が沸騰したか」という条件判断であり、もし条件が成立しなければ、「鍋のそばで待つ」という処理を「繰り返し」しているのであり、条件が成立すれば、「スパゲッティを入れる」という違う処理に分岐しているのである。おそらく、条件分岐、条件判断が難しいというのは、ワーキングメモリに負荷がかかるという意味のように思う。だが、条件分岐は、概念的にも内容的にも実践的にも、特別なものではない。

アルゴリズムを子どもに作らせるとしたら、誰かに守らせるルールを作りなさいという課題は作りやすいかもしれない。例えば、「あなたは図書館の館長さんです」と言ってから、貸し出しや返却の決まり、返却期限を過ぎても借りた本を返さない利用者への督促とペナルティなどに対するルールを決めさせる。「そのルールでは決まらないケースがある」など、担当者から苦情が来ないようにルールを作ってくださいと指示して、厳密に処理の手順を考えて記述することを促す。また、全体活動としてそのルールで万全かを検討する会を開き、問題点が見つかればルールを修正する。これはプログラミングを伴わないアルゴリズム作りの課題である。

4.2. 抽象化（abstraction）

「抽象化」は、しばしば誤解される用語である。すなわち、「抽象」なのだから、具体の反対の意味であり、事例が持っている本質的な性質や、複数の事例に共通する欠かせない性質を取り出すこと、概念化・一般化のことだと考えている人が多い。しかしながら、CT ではそういう意味ではない。

CT において、抽象化は、解決しようとする問題の複雑さや困難さを軽減することである。その軽減手段には、「分割」、「一般化」、「サブルーチン化」、「モデル化」などがある。

例えば「食事の用意をする」という課題を考えよう。この課題は、「ご飯を炊く」、「おかずを作る」、「みそ汁を温める」などを実行することで達成される。つまり、「食事の用意をする」というのは、「ご飯を炊く」、「おかずを作る」、「みそ汁を温める」といった一連の活動を一言で言い表した概念なのである。このような、いくつもの作業や内容や性質を一言のもとにまとめることが抽象化の一例である。このよ

うな、一連の手順を一言にまとめる作業は、サブルーチン化とも言える。すなわち、「ご飯を炊く」、「おかずを作る」、「みそ汁を温める」をひとまとめにして、「食事の用意をする」というサブルーチンを作るということである。サブルーチン化は、プログラミング言語 Scratch でいえば、「ブロックを作る」を使って、ひとつながりのブロックを、新たなブロックとして登録して、使えるようにすることである。

農業で、1 年に 2 回、同じ田畑で作物を作ることを「二毛作」という。「二毛作」という一言に、「農業で、1 年に 2 回、同じ田畑で作物を作ること」という意味を込めた抽象化を行っているのである。

掛け算は足し算の抽象化である。足し算を何度も繰り返すという手間を、掛け算は軽減しているからである。例えば、2×3 は、2＋2＋2 を表す。算数の授業は算数的なものの考え方を教えることが本来であるが、この「ものの考え方」が、子どもにはしばしば「自分の頭の中ではどう処理したか」という意味に受け取られている。そういう子どもは、「どうして 2×3 は 6 なのか」と聞かれたとき、「九九で 2×3 は 6 だから」と答える。もし、「2×3 は 2＋2＋2 のことだから」と答えてくれれば、この子どもは大変よく掛け算を理解していると言える。掛け算という演算を、足し算という、より基本的な演算の繰り返しとして説明できるというのは、CT 的にも算数的にも極めて適切な理解である。CT は、このような、より基本的な処理や要素に分解・分割し、それを組み合わせることで、元の問題を処理したり、事象を理解するという理解の仕方を、より意識的に身に着けてもらうことを期待しているのである。

たいていのパン屋さんは、自分たちで畑を持って、そこで小麦を栽培するところから始めたりはしない。質の良い小麦を相応の値段で買ってきて、パンを作っている。ラーメン屋さんも、麺は原料の小麦の栽培から始めたりしないで製麺業者から買い、スープのだしをとるための魚や鶏は市場から調達する。このような、自分たちで原材料を栽培したり飼育するところから始めずに、小麦粉や麺や鶏は買ってくるというのも、「おいしいパンを作る」とか「おいしいラーメンを作る」という、解決しようとする問題の複雑さや困難さを軽減する上で有効な手段であり、そうは見えないかもしれないが CT 的には抽象化なのである。

4.2.1. 分割・分解（decomposition）

「分割」・「分解」は、課題や活動を小さな課題や活動に分けることである。一連のつながりをもったもの（物語、授業など）は、「始め」「なか」「終わり」の 3 つや、「起」「承」「転」「結」の 4 つに分けることができる。時系列的な事象でない場合でも、生物を動物と植物に分けるとか、料理を和食、洋食、中華などに分けることが挙げられる。このとき、分割する狙いが、おおもとの課題を、より具体的に解決可能な小さな課題の集まりに変換することであれば、これら課題の分割は前述の「抽象化」でもある。分業は仕事の分割である。レストランであれば、料理を作る係、フロアで注文を取ったり配膳する係などに担当が分かれるが、これはレストラン運営という仕事を分割しているのであり、レストランの運営というタスクをより効率的に実行しているのである。

課題、仕事、あるいは動物植物などの分類や分割を行う背景には、元のものごとがそのままでは規模が大きすぎたり複雑すぎて、遂行不能であったり、理解困難であることがある。例えば、我々には「食事を作る」ということそのものはできない。我々にできることは、魚を焼いたり、みそ汁を作ったりすることである。だが、それらはあくまで「魚を焼く」とか「みそ汁を作る」であって、「食事を作る」ではない。「食事を作る」というのは、ご飯を炊いて、おかずを作り、提供するといった個々の作業の総体を指す。そのような意味で、課題や仕事の分解・分割は課題の達成に不可欠であると言える。

ものごとの理解という点でも、分けることは基本的な作業であると言える。何か理解したいとき、すなわち、それはどういうものであるか、あるいは、どうしてそうなったのかという問いに答えるには、とりあえずその対象をいくつかの目立つ事例に分けてみたり、どういうときにどういうことになるか場合分けをしたりする。生物なるものをまるごと理解することは一般には望めないので、とりあえず動く

もの（動物）と動かないもの（植物）に分けてみる。動物をさらに、哺乳類、爬虫類、鳥類などに分けてみる。哺乳類をさらに、人間、イヌ、ネコ、牛、豚、ライオンなどに分けてみる。哺乳類という抽象的な存在よりは、イヌや牛のほうがどのようなものであるか、例えば、体のサイズはだいたいどれぐらいか、それは何を食べるのかなどの質問には答えやすいだろう。さまざまな質問にきちんと答えられるのは、きちんと理解しているからである。

4.2.2. 一般化（generalization）

一般化は、複数の事例に共通する性質を抽出することであり、事例ごとに異なる性質を明らかにすることである。ここで性質というのは、「優しい」とかに限らず、「電気をよく通す」、「2 で割り切れる」、「羽根が生えている」、「甘い匂いにつられて集まる」など、広い意味で、事例や事物に当てはまる事柄を指す。

一辺の長さが 3 の正方形と、5 の正方形は、辺の長さが異なるだけで、それ以外は共通である。これら 2 つの正方形は、「一辺の長さが N の正方形」と一般化できる。このような正方形を描画するプログラムを書くとしよう。それぞれの辺の長さに対応した別個のプログラムを作るよりも、「一辺の長さが N の正方形」を描くプログラムを 1 本書いて、描画のたびに辺の長さ N の指定（パラメータ）だけ変えて実行することにはメリットが多い。なお、人によっては、一片の長さが 3 の正方形を描くプログラムを作ったら、次にそのプログラムをコピーして、辺の長さを 3 と指定している個所を 5 に書き換えることで、辺の長さが 5 の正方形を描くプログラムを作るかもしれない。このようなプログラムの再利用は労力をかけない賢い方法ではあるが、授業的には、一般化して得られた結果を、すなわち「一辺の長さが N の正方形」をプログラムして欲しいところなので、不満の残る方法ではある。だが、事例の共通点（正方形であること）と相違点（辺の長さ）を明らかにして、その結果を利用しているので、このプログラムを書いた人は一般化を行ったと言える。

一般化はパターンの発見と応用でもある。例えば、カレーライスと肉じゃがは違う料理だが、どちらも肉を炒めて、ジャガイモと一緒に煮込んで、味付けをするという意味では作り方は同じである。それぞれの料理で、玉ねぎをどれだけ炒めるか、ジャガイモのサイズをどうするか、などに違いはあるが、「肉を炒める」→「ジャガイモと一緒に煮込む」→「味付けをする」という基本パターンが認められよう。カレーも肉じゃがも、別個の料理と思うと作り方を覚えるのは大変そうだが、味付けが違うだけで基本パターンは同じであると考えれば、そんなに難しいことではないと思えないだろうか。なお、料理の世界では、カレーライスと牛丼はどちらも江戸のぶっかけ飯にルーツがあるとか、スパゲッティ・ナポリタンは洋風の鍋焼きうどんであるなど、それまでの料理や調理法が応用されて、新しい料理が生まれたと考えられる例は多い（澁川，2017）。

一般化は、ある作品や製品が、他にもどのようなことや場面で有用に使えるか、違う事例や用途を見出すことでもある。子どもが、ミュージックボックスを単なるプレイヤーとして使うのではなくアラームにするとか、LED を電池で点灯させるカードを作ったら、それを大掛かりにしたものが街灯であると気が付いた（Oliver, & Houchins, 2019）といった事例も一般化である。

4.3. 評価（evaluation）

ここで言う評価は、先生による成績評価ではなく、児童・生徒が、自分は課題を達成できたかを判定することである。PDCA サイクルにおける C（=Check）にあたる。

本論文で「評価」を CT の基本技能に含めた理由は、CT は問題解決活動である以上、評価は欠かせないためである。同時に、小中学校の授業では「まとめ」や「振り返り」という活動が行われており、自己評価を行うことは授業の一部を成していることもある。

評価は、学習を遊びに終わらせないためにも重要である。プログラムを書いたり、何か作品を作ったという場合、単に「先生に言われたとおりにやった」「できた」「楽しかった」では終わったのでは学習に

ならない。子どもが、自分はそれをどういうふうにして作ったのか、どういう手順でそれを行ったかという「振り返り」には、子どもに自分が学んだことを言語化させ意識させる役割がある。さらに一歩進んで、所期の目的はどれだけ達成できたか、達成できなかった部分があれば、それはどこであって、その原因はなんであるかを分析させて、適切な変更や改良を図ることは、プログラミングにおける「デバッグ」であり、プログラミング以外にも適用可能で有用な、問題解決の質を高める方法である。

何かの作品を作り上げる課題や、達成するゴールが設定されている場合は、子どもから評価活動を引き出す方法としては、「問題は解決したか？どうしてそうだと言えるか？」といった質問を投げかけることが考えられる（Oliver & Houchins, 2019）。

5．授業でのプログラミングの意識化と CT

プログラミング指導に関する小学校教員にとっての不安は、自分自身がプログラミングの知識・経験が不足していることや、授業での活用方法がわからないといったことが挙げられている（坂巻・福島, 2017）。だが、Scratch のようなプログラミング言語は、近年では豊富な印刷・映像資料が入手できるし、小学生でも簡単な導入的指導でプログラムを作れる。インターネットに安定的につながる環境であれば、プログラミング自体は大きな障害ではないと考えられる。むしろ問題は、プログラミングをいかにして教育活動として組み込むかであろう。

Scratch の特徴は、子どもでも、それなりに動くプログラムが容易に作れることにある。ここで「それなりに」というのは、プログラムを作る前に、どのようなプログラムを作るか十分に構想や準備をしていなくても、コンピュータの画面上に適当にブロックを並べていくと、Scratch のスプライトは、エラーで止まったりせず、少々変な動き方をしながらも動いてしまうという意味である。

初めてのプログラミングの授業では、プログラミングに慣れ親しんでもらい、その後の学習でプログラムを書くことを肯定的に受け止める態度に結び付けたいという狙いを設定するほうが現実的であるように思われる。ただし、それだけだと、「よくわからないが、プログラムを作って楽しかった」で終わってしまう危惧がある。そうならないために、授業中、子どもたちにどのような働きかけが考えられるだろうか。

それは、どういう仕組みでこのプログラムは動いているのか、という、仕組みに対する意識を持たせることであろう。もしプログラムが的確に書けなくても、プログラミング自体を考えさせること、どういう手順で動いているかまとめさせることは、プログラムというものに目を向けさせ、プログラミングという活動に対する基本的な認識を育てることに通じるのではないかと考える。ここで言うプログラミングに対する基本的な認識を定式化したものが CT でありプログラミング的思考である。これまでに述べた、「抽象化」や「アルゴリズム」などをそれらとして意識しなくても、自分の意識を働かせて、プログラムには仕組みというものがあるとか、それはこのプログラムではどのようになっているだろうかという思考が洗練されていった先に、「抽象化」や「アルゴリズム」などの CT の概念がある。

6．Computational Thinking の課題

CT の課題とされることのうち、特に以下の 3 つを挙げる。

1 つは、これまでも述べたが、CT の定義について、幅広く合意され確立されたものがないことである。その原因は、定義を提唱する研究者・実務家が、教育系の人か CS 系の人かというバックグラウンドの違いと、その人の教育の意義や狙いに対する考え方（教育に対する持論）が多岐にわたることが挙げられる。おそらく、定義の問題は今後も議論が続くであろう。だが、大枠では、論理的合理的思考力に根差した問題解決能力ということではないかと思われる。

2 つ目は、カリキュラムをどのようにするのが適切であるかについても、幅広く合意され確立されたものがないことである。この点は、CT の定義が確

立されていないことと連動している。CT が何であるかが確定していないと、何を教えなくてはならないか、教育目標をどう設定するかも確定しないことになる。当面は、学習指導要領を踏まえつつ、各学校や児童・生徒の現状に合わせて、さまざまなカリキュラム提案を参考に模索が続くだろう。

3 つ目は、成績や理解度の評価基準・規準についても幅広く合意され確立されたものがないことである。CT の定義が定まっていない状態で評価基準・規準を定めることは容易ではないが、この状況は日本に限ったことではない。現状では、CT の成績や理解度の評価は行われていなかったり、個々の授業実践ごとに規準が設定されている。そもそも、問題解決活動はあらゆる分野・場面で現れる、一般的・普遍的な活動なので、それを系統的な方法論として規定しても、個々の、具体的な問題や分野に固有な知識や情報が不足していては、具体的な問題の解決につながりにくい。その意味で、評価を前提としないが、全体の教科に通じている「総合的な学習の時間」でプログラミングを体験させた後に、算数・数学や理科などの各教科でプログラミングを含む授業を行うという流れが適当であろう。

文献

赤堀侃司（2018）プログラミング教育の考え方とすぐに使える教材集．ジャムハウス

Barr, V. & Stephenson, C. (2011) Bringing computational thinking to K-12: What is involved and what is the role of the computer science education community? *ACM Inroads*, **2**(1), 48-54.

Computing at School Working Group (2012) *Computer Science: A Curriculum for Schools*. http://www.computingatschool.org.uk

Doleck,T., Bazelais,P., Lemay,D.J., Saxena,A., Basnet,R.B. (2017) Algorithmic thinking, cooperativity, creativity, critical thinking, and problem solving: Exploring the relationship between computational thinking skills and academic performance. *Journal of Computers in Education*, 4, 355-369.

diSessa, A. A. (2000). *Changing minds: Computers, learning, and literacy*. Cambridge: MIT Press

Grover,S. & Pea,R. (2013) Computational thinking in K-12: A review of the state of the field. *Educational Researcher*, **42**(1), 38-43.

Kernighan, B.W. (2011) *D is for Digital: What a well-informed person should know about computers and communications*.（久野靖 訳（2013）ディジタル作法　カーニハン先生の「情報」教室．オーム社）

Korkmaz,O., Cakir,R., Yasar Ozden,M. (2017) A validity and reliability study of the computational thinking scales (CTS). *Computers in Human Behavior*, **72**, 558-569.

Lee,I., Martin,F., Denner,J., Coulter,B., Allan,W., Erickson,J., Malyn-Smith,J., & Wermer,L., (2011) Computational thinking for youth in practice. *ACM Inroads*, **2**(1), 33-37.

Lu,J.J., & Fletcher, G.H.L.(2009) Thinking about computational thinking. In *Proceedings of the 40th ACM Technical Symposium on Computer Science Education (SIGCSE'09)* (pp.260-264). Chattanooga, Tennessee, USA.

Lye,S.Y., Koh, J.H.L. (2014) Review of teaching and learning of computational thinking through programming: What is next for K-12? *Computers in Human Behavior*, **41**, 51-61.

増原英彦（2006）問題の解き方．川合慧 編（2006）情報．東京大学出版会

Mishra,P. & Yadav,A., & the Deep-Play Research Group (2013) Rethinking technology & creativity in the 21st century: Of art and algorithms, *TechTrends*, **57**(3), 10-14.

文部科学省（2017）小学校学習指導要領

文部科学省（2018）小学校プログラミング教育の手引（第二版）

National Research Council. (2012) *A framework for K-12 science education: Practices, crosscutting concepts and core ideas*. Washington, DC: National Academies Press.

Oliver,K., & Houchins,J.K.(2019) Evidence of computational thinking from circuitry projects in the after-school makerspace. In *Proceedings of EdMedia+Innovate Learning 2019* (pp.364-369). Amsterdam, Netherlands.

尾崎　光，伊藤陽介（2017）小学校におけるプログラミング教育実践上の課題．鳴門教育大学情報教育ジャーナル，**15**，31-35.

Papert, S. (1980) *Mindstorms: Children, Computers, and Powerful Ideas*. New York: Basic Books.

Papert, S. (1993) *The Children's Machine: Rethinking school in*

the Age of the Computer. New York: Basic Books.

Polya, G. (1957) *How to solve it: A new aspects of mathematical method*. Princeton: Princeton University Press.

Resnick,M., Maloney,J.,Monroy-Hernandez,A., Rusk,N., Eastmond,E., Brennan,K., Millner,A., Rosenbaum,E., Silver,J., Silverman,B., & Kafai,Y. (2009) Scratch: Programming for all. *Communications of the ACM*, **52**(11), 60-67.

Rodger,S.H., Hayes,J., Lezin,G., Qin,H., Nelson,D., Tucker,R. (2009) Engaging middle school teachers and students with Alice in a diverse set of subjects. In *Proceedings of the 40th ACM Technical Symposium on Computer Science Education (SIGCSE'09)* (pp.271-275). Chattanooga, Tennessee, USA.

Royal Society. (2012) *Shut down or restart: The way forward for computing in UK schools*. http://royalsociety.org/education/policy/computing-in-schools/report/

坂巻若菜，福島健介（2017）授業実践から考える小学校におけるプログラミング教育の課題・方向性．2017 PC Conference，151-154.

澁川祐子（2017）オムライスの秘密　メロンパンの謎 人気メニュー誕生ものがたり．新潮社

Voogt, J., Fisser, P., Good, J., Mishra, P., & Yadav, A. (2015). Computational thinking in compulsory education: Towards an agenda for research and practice. *Education and Information Technologies*, **20**(4), 715-728.

Werner,L., Denner,J., Campe,S. (2012) The fairy performance assessment: Measuring computational thinking in middle school. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education (SIGCSE '12)* (pp.215-220). Raleigh, North Carolina, USA.

Wolz,U., Stone,M., Pearson,K., Pulimood,S., Switzer,M. (2011) Computational thinking and expository writing in the middle school. *ACM transactions on Computing Education*, **11**(2), article 9.

Yadav,A., Stephenson,C., & Hong,H. (2017) Computational thinking for teacher education. *Communications of the ACM*, **60**(4), 55-62.