Bulletin of Aichi Univ. of Education, 67 (Educational Sciences), pp. 29-32, March, 2018

# プログラミング教育の低年齢化に伴う大学での授業について

松永　豊
情報教育講座

About the Class at the University Accompanied by the Low Age of Programming Education

Yutaka MATSUNAGA
*Department of Information Sciences, Aichi University of Education, Kariya 448-8542, Japan*

## １．はじめに

近年，プログラミング教育の低年齢化が叫ばれている。言うまでもなくプログラミングとはコンピュータに所望の操作を命令することであり，その命令には人工の言語を扱うことになる。自然言語と同様さまざまなキーワードを組み合わせることで複雑な制御を可能としている。命令群の組み合わせにより実行結果が変わる，文法が存在する，など，自然言語と同様の特徴も有するが，一般に予約語（リザーブキーワード）の数は自然言語と比べると圧倒的に少ないことが特徴である。それゆえ，プログラミング能力は暗記的な能力よりも複雑な思考を伴う能力が重要と考えている。

本学，情報科学コースにおいても従来からプログラミング教育を行っている。本学は教員養成大学であるが，情報科学コースはゼロ免課程（教育職員免許状の取得を卒業要件としない課程の通称）からスタートしており，現代学芸課程（ゼロ免課程）と教員養成課程の学生の両方を指導するようになってからも，以下のようなやや特殊なコースの特徴を有していた。

- 情報教育講座教員は現代学芸課程（ゼロ免課程）の情報科学コースと，教員養成課程の初等情報，中等情報の３つのコースの学生の育成を主に行っていたこと
- 工学系出身である情報教育講座教員の割合が高かったこと
- 長きにわたってゼロ免コースが存在したため，企業に就職する学生の割合が高かったこと
- 教員養成課程中等専攻では高校（情報）の免許に関する課程認定を受けていたこと
- 現代学芸課程と教員養成課程の２種類のコースを両輪としてカリキュラムが構築できたこと

このような要因により，教育大学としてはやや異例なほど重厚なプログラミング教育を行う環境が整っていた。（表１参照）

●必須　○選択

| 授業名 | 使用言語 | 現代学芸 | 教員養成 |
|---|---|---|---|
| プログラミング１ | PEN, C++ | ● | ● |
| プログラミング２ | C++ | ● | ○ |
| プログラミング３ | C++ | ● | ○ |
| プログラミング方法論 | C# | ○ | ○ |

表１　プログラミング関連授業

残念ながら現代学芸課程が廃止されてしまうため，今後のカリキュラムは全体的に教員養成向けにシフトすることになる。しかしながら，２コース両輪で授業提供をしてきたさまざまな経験から，今まで蓄積されたノウハウを活用して授業運営可能と考えている。

さて，冒頭の話題に戻すが，近年，プログラミング教育の低年齢化が大幅に進んでいる。理由は色々とあるが，デジタル技術の驚異的な進展による社会構造の変革，いわゆる第四次産業革命の存在は無視できないだろう[1]。今後，多くの職業が人工知能（AI）に置き換わると言われており，現在の子供たちが将来大人になるころには存在しなくなる職業や業種も多くなるかもしれない。しかしながら，逆に情報産業を支える人材は不足しているため，今後の就職活動等ではプログラミング能力は有利な能力となるかもしれない。

そこで本研究では，工学的観点からのプログラミングと教育的観点からのプログラミングを比較し，工学的観点を生かした教育手法やカリキュラムの提案，および，さまざまな問題点や解決方法の議論を目的とする。

## ２．ブロック型プログラミング言語の利点欠点

周知の通り，次期学習指導要領において 2020 年度から小学校プログラミング教育が必修となるため，現在，さまざまな観点から低学年プログラミング教育の研究が模索されている。特に注目されているのがブロックパーツを使ったビジュアルプログラミング開発環

境である。

筆者は現在小学校プログラミング教育の研究にも携わっており，実際に小学校でプログラミング教育を行っているが，そこでも Scratch [2]を使っており，子供たちが楽しそうに課題作成に取り組んでいる姿を見るたびに，さすがに低学年学習者にも配慮して開発されただけのことはあるな，と感心している。

一方で Scratch のようなブロック式の開発環境は，テキストプログラミング既修得が使うと必ずしも楽ではないことも事実である。

例えば，５＋３の足し算を考えてみよう。Scratch に演算の中には以下の足し算ブロックがある。

**図 1　足し算ブロック**

あとは丸の中に数字を入力すれば終わりである。難しい点は特にない。

**図 2　足し算　５＋３の例**

次に５＋３＋８＋７を考えてみる。以下の例は答えの一つであるが，図 1 のブロックを 3 つ組み合わせてある。

**図 3　足し算　５＋３＋８＋７の例**

同様に２５－３－７－８を考えてみる。足し算ブロック同様引き算ブロックもあるので組み合わせると以下の答えが導き出せる。

**図 4　引き算　２５－３－７－８の例（正解）**

しかしながら，以下の組み合わせは間違っている。

**図 5　引き算　２５－３－７－８　？（不正解）**

無論，図 4 は(((25-3)-8)-7)であり，

図 5 は((25-3)-(8-7))**である**

ことが理由である。すなわち，ブロックを使った演算の場合，大量のブロックをマウス操作しなければいけないことや優先順位がシビアになることで面倒に感じることも多いかもしれない。この傾向はテキストプログラミング言語習得者ほど顕著になることが予想される。言うまでもなくテキストプログラミングではエディタに　25-3-8-7　と入力するだけである。なぜこんなにもブロックプログラミングでは煩わしくなるのだろうか。逆に言えば，テキストプログラミング言語ではなぜここまでスマートになるのだろうか。この疑問は，例え Scratch 等で初めてプログラミングを学ぶ人にとっても知っておいてよい要素と考えられる。ましてや，教育する立場になる人であれば当然知っておくべきである。（子供から質問されるかもしれない）

答えは，テキストエディタ自体がそもそも膨大な状態を扱っていることに他ならないからである。先述の通り，プログラミングの本質は組み合わせである。少ない予約語や記号や数字等を，例えば，if や for や while や 25 や 3 や＋や＝をどういう順番で並べるかがプログラミングの本質である。ブロック型プログラミング言語でもこれは変わらないのであるが，エディタはブロックそのものを細切れで表現してしまう。

別の言い方をしてみよう。1 年は 12 か月あるが，ブロック型プログラミング言語はこれに名前を付けたような感じである。例えば，1 月には Jan，2 月には Feb，3 月には Mar，…，12 月には Dec と付けたと考える。すると，その世界には 12 種類のデータしか存在しないことになる。一方でエディタは 0,1,2,…,9 の記号のまま管理しているようなものである。例えば 10 であれば「1」「0」のように管理している。当然，1～12 を扱うことはできるが，同時に 13 や 47 のような本来使用不可能な範囲まで表現可能になってしまう。これを是（自由度が高い）とするか非（不要なものを扱う羽目になる）かで評価は分かれるが，エディタ式の方は圧倒的に多くの情報を扱っていることに他ならない。当然，不正なデータは排除しなければならないがそれを行っているのがコンパイラともいえる。

すなわち，最初から制限をかけて入力自体をできなくするか，入力自体は許して整合性が取れないものを排除するかの違いともいえる。つまり，低学年向けプログラミング言語の最大の特徴である「素人にも扱いやすい」という点は，自由度とバーター（自由度の犠牲）と換言することも可能だろう。ブロック型プログラミング言語をテキストプログラミング言語と比較した場合のデメリットは他にもある。先にデメリットを知ってしまえば逆にブロック型プログラミング言語を使う際にも参考になるので，大学の授業ではこの点も教えるべきだと考えている。

## ３．プログラミングで扱う題材について

C++，C#，JAVA など，自由度の高いプログラミング言語を扱う場合，究極的な目標は完全に自由なソフトの開発となる。無論，限られた時間内に完成させる

などの制約があるので，完全なフリー課題というわけにはいかないことも多いが，穴埋め問題のような，ドリル課題のようなものだけではつまらないので，ある程度の自由度を残した課題を最終課題にしたいと考えて組み立てられたカリキュラムは多いだろう。

前述の通り，現代学芸課程のコース向けには複数のプログラミングの授業が開かれていたが，特にプログラミング３とプログラミング方法論では自由度の高い課題を課していた。それぞれ，C++コンソールアプリケーション作成，Visual Studio を用いた C# Windows アプリケーションの作成と開発環境や完成度は全く異なるが，どちらも最終課題のテーマ自体は自由である。学生は，まず作成したいソフトについて仕様書を作成してチェックを受け，仕様書が合格した者からソースコード作成に取り組み，完成したらプログラムを個別面談という方法でチェックを受ける，という手順となる[3]。

さて，学生には自由な発想でプログラミングを組んでもらうわけだが，結果的には扱う題材（ジャンル）はゲームとなることが多い。無論，授業中の練習課題で扱うテーマがじゃんけんゲームや落書きツールなど遊び要素が多いことや，指導者本人（筆者）が説明の際にカードゲームやオセロ等をネタに使うことが多いことが要因になっていることは否めないが，一応，最終課題の説明の際にはゲーム以外のジャンルであるツールや学習支援ソフトなども紹介しており，最終的には学生の裁量に任せているので，プログラミングの自由課題を完全にフリーハンドにするとゲームに落ち着く確率が高いものなのかもしれない。


**図 6　学生が作成したソフトの例**

現代学芸課程情報科学コースの学生は大半が企業に就職するわけだが，業務にプログラミングが伴うような企業に就職する場合，どのような職種であろうと一般的なプログラミング能力が役に立つと考えられる。就職先でもゲームを作ることなる学生は極めて少ないだろうが，ゲーム作りのときにノウハウが役立つことは大いにあり得る。プログラミングをする際に人によっては壁になり得る概念，例えば，配列の扱い，再帰構造，データ構造（構造体，クラス等）などは，ゲーム作成では勿論のこと，どの分野でも使われるだろう。（なお，さらに壁になることが多い概念としてはポインタがあるが，本学の授業ではあえて扱ってはいない。）

| 壁になり得る要素 | 説明・授業での扱い等 |
|---|---|
| 配列 | ● 繰り返し構造の変数<br>● 授業では使用が必須<br>● 配列を使わずに済む課題は極めて小さなものか簡単なものになるので，含まれない場合は仕様書提出の段階ではねられる。 |
| 再起構造 | ● 自分自身を呼び出す関数<br>● アルゴリズム解説書でもよく見かける<br>● 戦略系ゲームを作る場合やマインスイーパーのような塗りつぶし系の処理をする際には重宝 |
| データ構造（構造体，クラス） | ● さまざまな型のデータをまとめる技術<br>● 引数で使用する場合等も便利<br>● クラスはオブジェクト指向においては必要不可欠 |

**表 1　壁になり得る概念等**

一方で教育を目指す学生に求められるものは少々異なるかもしれない。ゲーム作り等はあくまでも趣味の延長になるかもしれないので，扱うべきテーマを限定することが必要になる。

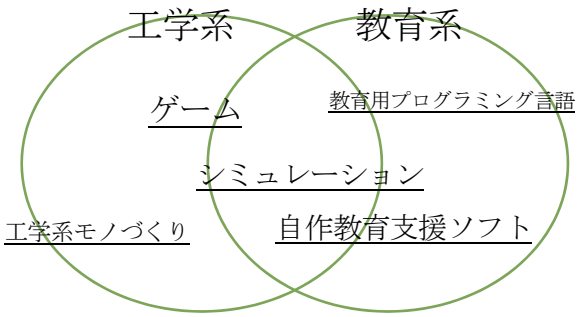


**図 7　題材の選定について**

教員自体のプログラミング能力が上がることですぐに思いつくメリットは授業で使用する教材を自作できる可能性が向上することである。この場合，簡単なソフトでもいいのでなるべく予備知識なしで作りたいのであれば，ブロック型プログラミング言語は有効であると考えられる。ただし，逆に教材開発にはまり，いろいろなソフトが作成したくなった場合は，前章で述べたような不満が出てくる可能性は大いにあり得る。

前章で述べたような不満が解消されているビジュアルプログラミング開発環境が登場すればよいが，最終的にはテキストプログラミング言語を学ぶのが近道になる可能性も捨てきれない。

教員養成課程学生向けのプログラミング学習教材としてはシミュレーションも選択肢に入るだろう [4] [5]。この場合，敢えてプログラミングを使わずに Excel でシミュレーションをする方法もある。Excel 上で VBA 等も使わずに行うシミュレーションであるが，セル間の関係式等を工夫すれば論理的思考力向上の課題にはなり得る。(純粋にプログラミング教育となるかといえば微妙ではあるが…)

## 4．オブジェクト指向に関して

次にオブジェクト指向に関しても言及をしておく。低学年向けプログラミング開発環境は，オブジェクト指向をベースとしたものが多い。例えば Scratch もそうである。子供にプログラミングを教える際にアルゴリズムから入るよりもイメージしやすいモノから入るほうがスムーズになると思われるので，オブジェクト指向自体にはまったく異論はない。

しかしながら，オブジェクト指向に拘り過ぎた場合はプログラミングが煩雑になることも事実である。例えば，別の個体を操作する場合はメッセージを飛ばすなどの方法で処理せざるを得ない。

現実世界ではこれが自然である。例えばサッカーチームの監督はサッカー選手に指示を出すことはできるが，実際にサッカー選手が移動したり球を蹴ったりするのは本人の意思だからである。しかし，これに拘り過ぎると将棋等のプログラムでは圧倒的に煩雑になる。将棋では差し手がすべてのコマを自分の意志で動かす。歩に「前に進んでくれ」と命令して，歩が自らの意思で前に進むわけではない。ゲーム上やコンピュータ上の世界はあくまでも仮想世界であることのほうが便利な場合もあるという意味である。

すなわち，コンピュータのプログラミングを考えた場合，入り口は現実社会との関連性が深いほうが望ましいが，情報処理機械に処理をさせると割り切る場合には少々面倒になるかもしれないことを忘れてはいけないと感じている。

ブロック型プログラミング言語に話を絞るととしたら，同じ MIT の開発環境で比較した場合，App Inventor 2 ぐらいが限界で[6]，Scratch はストイックすぎる気がしなくもない。

## 4．まとめ

以上，本研究ではプログラミング教育の低年齢化に関して，ブロック型プログラミング言語をはじめとするビジュアルプログラミング言語が使用されている現状を説明した。また，それに伴い大学教育でのプログラミング教育についての提唱を行った。具体的にはブロック型プログラミング言語そのもののメリットデメリットを再認識が重要と考える。また，課題作成においては扱うテーマを再検討すべきであると述べた。また，オブジェクト指向プログラミングについても少し言及した。

次期学習指導要領の改訂で 2020 年度から小学校でプログラミング教育が必修になるなど，プログラミング教育の低年齢化を止めることはもはや不可能であるし，早い段階で子供に ICT 機器を触らせることに問題がないのであれば，早い段階でコンピュータの仕組みやプログラミングを理解することは極めて重要であると考えている。

一方で，能率的なプログラミング作成においては現在のブロック型プログラミング言語では不十分であるとも考えている。ビジュアルプログラミング言語の性能向上に舵を取るべきか，ビジュアルプログラミングからテキストプログラミング言語のシームレスな移行手段を確立させるべきなのかは，しばらく議論が続きそうであると考えている。

## 参考文献

[1] 第 4 次産業革命がもたらす 変革 - 総務省 http://www.soumu.go.jp/johotsusintokei/whitepaper/ja/h29/pdf/n3100000.pdf

[2] https://scratch.mit.edu/

[3] プログラミング演習授業支援システムの開発，愛知教育大学研究報告．教育科学編，愛知教育大学，59:169-174, 2010-03

[4] 自作教材ソフトを用いたシミュレーション演習授業,愛知教育大学研究報告．教育科学編，愛知教育大学，61:201-205, 2012-03

[5] Excel を用いたシミュレーション演習授業，愛知教育大学研究報告 教育科学編，愛知教育大学，60:187-190, 2011-03

[6] http://ai2.appinventor.mit.edu/

（2017 年 11 月 30 日 受理）