Journal of Kochi Rehabilitation Institute 19 (1)：1－6 (2017)　　ISSN 1345-5648

プログラミング教育の事例について

竹島　卓

平成29年度　高知リハビリテーション学院紀要（平成29年９月）第19巻１号　別刷

金沢工業大学基礎教育部

総説

プログラミング教育の事例について

竹島　卓

A note on programming education

Taku Takeshima

要　旨

文部科学省の「教育の情報化の推進」の中に「プログラミング教育」が掲げられ，2020年から小学校でプログラミング教育が必修となる．情報処理を専門としない大学の教育でも，「情報活用能力」が求められており，その素地としてプログラミングの基礎素養は必要となる．本稿では，情報系学科のプログラミング授業に筆者が10年間携わった経験からプログラミング教育の難しさを紹介する．まず，Infomation Technologyが現在社会にとって不可欠の技術であることを述べ，その核心としてInfomation Technologyを担っているコンピュータが動作するためのプログラムを作ること，すなわちプログラミングが現在およびこれからの社会にとって重要であることを説く．しかしながら，プログラミングの特性からそれができる人とできない人とが顕著に，しかも他の教科とは無相関に分かれており，教育の難しさをもたらしている．そこで，できない人の特徴を特定し，その克服の試みを紹介する．最後に情報を専門としない学科でのプログラミング教育において考慮すべき事項に言及した．

キーワード：Infomation Technology，プログラミング，大学教育

【はじめに】

文部科学省は平成14年６月に「情報教育の実践と学校の情報化」という文書を公表し，情報教育の重要性を指摘した．文部科学省の「教育の情報化の推進」の中には「プログラミング教育」が掲げられ，小中高の学校教育の中でプログラミング教育を行うことが謳われており，2020年には小学校からプログラミング教育が必修化される．その背景には，深刻なわが国の Infomation Technology （以下， IT）人材不足の予測がある．つまり，世界的に IT 需要は拡大するにも関わらず，少子化による人口減少時代のわが国では，2030年の人材不足規模が約59万人（中位シナリオ）に達するという[1)]．

文部科学省の提唱・推進する「情報活用能力」は，「情報活用の実践力」，「情報の科学的な理解」，「情報社会に参画する態度」の３要素から成るとされているが，これらの能力の下地としてプログラミングに対する基礎素養が求められる．大学においても，この「情報活用能力」の教育実践が求められており，プログラミングを専門としない学科・専攻でもプログラミングをどの程度，どのように教えるかが問われている．

筆者は平成19年からこれまでの10年間，北陸の工科系大学に在籍し，情報系学科の初年次から2年次の学生に対して， C 言語でプログラミングを教えて来た．その経験から感じたプログラミング教育の難しさと克服の取り組みをここに紹介したい．

金沢工業大学基礎教育部
Practical Engineering Education program, Academic Foundations Programs, Kanazawa Institute of Tecnology

【プログラミングは何のために役立つ？】

ITは，すでに人間社会にとってなくてはならないものとなっている．食料やエネルギーの生産，生活用具や機械の製造，建築，人・物の移動と流通，社会インフラの整備，医療，教育，スポーツ・娯楽など，あらゆる人間活動の中でIT化が進んでいる．

さらにここ数年の間に人工知能（Artificial Intelligence：以下，AI）の研究が目覚しい進展を見せ，AIの能力が急速に高度化した．その結果，囲碁・将棋のようなボードゲームの世界では，人間はもはやAIに勝つことができなくなっている．

このようなITの核心にあるものがコンピュータである．コンピュータはプログラムと呼ぶ情報を内蔵(記憶)し，そのプログラムに書かれた命令に従って忠実に情報の処理を行う．入力装置を通じて情報をコンピュータの外部から読み取ることや，処理した情報をコンピュータの外部にある出力装置に書き出すことができる．つまり，コンピュータは外界の状況に応じて必要な処理をし，外界に働きかけることができる．

そのためコンピュータは，人間の脳が感覚器と神経を通して送られてくる情報を処理して，神経を通じて効果器に必要な情報を送って，筋肉を動かすことと同様の働きをする．

コンピュータを動かすための命令は200種類程度であるが，その組み合わせで，理論的には任意の処理が（どんな複雑なことであろうと）可能になっている．

もちろん，感覚器や効果器に相当する適切な入力・出力装置があることは前提である．目的の動作(情報の処理)に適う命令の組み合わせ(プログラム)を作ることがプログラミングという作業である．

【プログラミングとはなにか？】

１．プログラミング

プログラミングとは，予め定められた命令のセットを用いて，入力された情報から出力するべき情報を計算する命令の組み合わせ列を作成する作業である．このとき，目的は要求仕様という形で通常は日本語などの自然言語で与えられる．

前節ではコンピュータ本体が備えている命令のセット(機械語命令のセット)を想定して，プログラミングを説明したが，実際に機械語でプログラミングすることは人間にとって負担が大きすぎるため，個々の命令とその組み合わせ方が人間にも理解し易い，プログラミング言語が考案されており，それを用いてプログラミングすることが普通である．例えば，古くはFortran，COBOL，Cや最近ではJava，Pythonなどが挙げられる．

２．プログラミング言語の要素

プログラムの基本的構成物にはつぎのものがあり，それを組み合わせてプログラムが作られる．組み合わせ方を規定するものはプログラミング言語の文法である．

1)変数　...状態を記憶する仕掛け
2)式の計算　...基本的演算，状態を変更するわけではない
3)代入　...状態を変更する操作
4)３つの制御構造　...つぎに実行する命令の決め方
（１)逐次処理（命令は順に実行される）
（２)判断分岐（条件の成立，不成立に従って，つぎに実行する命令が変る）
（３)繰り返し（条件が成立している間繰り返す）

プログラムはこれらの要素をプログラミング言語の文法に従って記述したものである．文法に従っているプログラムにはその意味(コンピュータの動作)が定義される．

３．プログラムを作る作業

さて，目的を実現するプログラムは無限に有り得るが，計算効率や論理の明快さ，表現の簡潔さ，メンテナンスの容易さなどのさまざまな観点から，吟味され評価される．

プログラムを作るということは，自然言語で表現

される要求仕様を段階的に分解して行き，最終的にはプログラミング言語が提供している文に翻訳して，プログラミング言語の文章として再構成する作業である．

この作業は，組み合わせ的であり，各部分の時間的順序関係やその他の論理的関係を常に把握して実施しなければならない．

一旦要求仕様が定まると，プログラムに与えられる入力と出力の関係は論理的な関係であり，物理的な制約を受けないため，設計の自由度は大きい．しかし，その代わりにコンピュータの動作(状態の変化)について物理的な直観に頼ることはできず，そういう意味で極めて論理的で知的な作業である．

【プログラミングの向き不向き】

「世の中には２種類の人間がいる(しかいない！)．プログラミングができる者とできない者とである．」というジョークがあるほど，プログラミングは向き不向きがある．よく，「論理的思考を鍛えるためには数学が良い．」と言われる．また，最近ではあまり聞かなくなったが，「日本語は論理的でなく英語は論理的である．」とも言われていた．

筆者が現在所属する大学の情報系学科で，1年次のプログラミング科目の成績と，入学時の数学力，英語力との相関を調べた結果がある．それによると，まったく無相関であった．「プログラミングの学力は数学，英語の学力とは無関係である」との見解が支持される結果となっている．一方，国語力との関連が考えられるが，国語力の調査データはなく検証ができていない．

この現象の説明の一つとして，このデータの母集団となった学生達にとってのそれまでの『数学』や『英語』の学びでは，つぎの関係式が成り立っていたのではないかと推察している．

『数学』力＝計算力および公式の機械的適用力
『数学』力≠論理的思考力
『英語』力＝記憶力および反射的応答力
『英語』力≠論理的表現力

実は数学についてはより強い無関係性が言えそうである．

プログラミングができる人の思考形態をプログラム的思考と呼ぶことにしよう．するとこのデータから直接見て取れることは，プログラム的思考は，『数学』ができない人でもできているし，『数学』ができる人でもプログラミングはできない人もいる．であるが，実は，この学生たちに限らず一般の意味で，数学ができなくても，プログラミングの達人はいるし，数学者でもプログラミングが全くできない人はいくらでもいる，ということも間違いなく真実である．ここで，「プログラム的思考」は「アルゴリズム思考」と言ってもよいだろう．

【プログラミングができない学生の特徴】

１．３つの特徴

そもそも真面目に勉学に取り組まない学生は別として，プログラミングが不得手な学生では，つぎの３つが顕著である．

１）言葉に無頓着，注意深く無い

「,」,「;」,「.」,「:」の区別や「( )」,「{}」,「[ ]」の区別をしない．これは，繰り返し注意することで解決するが，つぎは改善が容易ではない．

使っている変数の名前が，それが表しているものの実態とそぐわない．また，意味の無い符号に過ぎない命名が多い．これには語彙が少ないことも関与していると思われる．

２）プログラムのダイナミクスを理解せず，プログラム実行による状態(変数の値の変化)が追えない

プログラムを作る時点とプログラムが実行される時点・環境の違いを認識せずに行う．例えば，変数の名前が goukei のとき，最初から goukei となるべき値を設定して一切変更をしないプログラムを書いてしまう．また，実行される状況に応じて結果は変わるべきだが，常に同じ結果しか与えないプログラムを書いてしまう．外界はそのプログラムにとって都合のよいデータしか入力しないという，予定調和型のプログラムを書いてしまうことも多い．

３）数学の計算の慣習をプログラミングに持ち込む

とくに C 言語では代入を「←」ではなく「=」で表すことから，代入と等式を混同する．つまり，プログラムの計算式が式として記憶されていると誤認している．そのため，下記のような誤りが生じる．

（例）

x ← y＋1

とした後で，

y ← 2

とすれば，x は３になる，と思い込んでしまう．

２．『数学』の弊害

とくに前述した３）と２）については，母集団になった学生達の受けた現行の教育での『数学』の弊害が現れていると考える．

学生たちは，「答え（正解）が必ずある」「間違えずに機械的に計算すれば必ず正解が得られる」ものと考えている．そのため，答えは，創り出すものではなく，static に存在しており，在るものを見付けて持って来ればよいと，習慣付けられているのではないかと，筆者は疑っている．

例１）$\Sigma_{i=1}^{n} i^2 = \frac{1}{6} n(n+1)(2n+1)$

n=3 のときの答えは $\frac{1}{6}3(3+1)(2\cdot 3+1)=14$

と計算して得られる．最後の14は何かが変化して得られたものではない．

実際，要素を繰り返し加えることで，最終的な総和14を得ることを指示したプログラミングの課題でも，この例のように計算式で結果を求めた学生がいた．

一方，プログラミングでは，「答えはあるかどうか，そこに行き着くかどうかも分からない」，「仮の答えを作って，それを修正しながら答え（らしきもの）に近づいて行く」．

このため，変数の値は常に「暫定値」であり，答えは創り出すものとして，「全てを動的（dynamic）に」捉える必要がある．

例２）$\Sigma_{i=1}^{n} i^2 = 1^2+2^2+ \ldots + n^2$

n=3の答えは，仮の答え s を暫定的に０にしておいて，それに３回，回数に応じて値を加えていく．

最初（i=1）には 1^2= 1を加えて，s=1 となる．まだ，暫定値．

２回目（i=2）には 2^2= 4を加えて，s=5 となる．まだ，暫定値．

３回目（i=3）には 3^2= 9を加えて，s=14となる．

ここで初めて，s が真の答えになった．

実際 s を合計値と呼ぶことが多いが，s が合計値14であるのは，計算手続きが終了した丁度そのときだけである．計算手続きが終了しないうちは，経過的な暫定値に過ぎない．しかし，多くの場合がそうであるように，s はいつかそうなるに過ぎない呼び名である「sum」，つまり合計，を意味するように「sum」のイニシャル s が使われている．

【プログラムの特徴】

プログラムの重要な特徴としてつぎの３つがある．当たり前だと思わせず意識させることが初心者には必要である．

１．プログラムを書く（作る）時点・環境と，それが実行される時点・環境は異なる

２．プログラムはコンピュータ（状態機械）が実行する．

変数はプログラムの進行とともに値が変化するため，実行中の変数の値は常に仮の値（暫定値）にすぎない．

３．プログラムの命令は必ず実行される．（どこかの法律のように「骨抜き」にはならない）

【プログラミング落ちこぼれの対策】

１．課外時間の活用

プログラミング教育の経験から，上述した特徴を持つ学生がいることが分かっている．

そのことを踏まえて，誤認識や認識不足の可能性を疑い対応する必要があるが，問題を抱える学生に個別に対応するのでなければ，効果は薄い．かといっ

て，授業中は問題のない多くの学生の成長に注力したい．従って，課外の時間を個別対応に割くことになる．

２．課題の量と質

問題のある無しに拘わらず次のトレードオフの問題が常に存在する．

１）学生は成績に寄与しなければ自分からプログラム練習をすることはない
２）プログラミング課題は教員の採点負荷が大きい
３）実践的なプログラミング力を培うには，実用的な課題が望ましい
４）実用的な課題が望ましいが，妥当な時間で完成する必要がある

３．ティーチングアシスタントの活用と教え合いの推奨

これらの調整の一助として，ティーチングアシスタントの活用と学生同士の教え合いを推奨した．教え合いは，先に課題が解けた者がまだ解けていない学生の支援をするという方式である．ネズミ講式に解決者が増加することを期待するが，実際はそうはならず，あちこちから教員やティーチングアシスタントにお呼びが掛かるのが実情である．それでも積極的に課外時間を活用する学生は伸びが期待できる．

【プログラミングの障壁】

C 言語の場合，つぎのところで躓きが生じる．

１．条件そのものの記述（かつ，または，～でないの組み合わせ）
２．条件文の記述（if ～ else の入れ子構造）
３．繰り返し（決まった回数の繰り返しはできる）条件を満たさなくなるまで繰り返す形式は苦手
４．配列の概念，配列要素に関する繰り返し
５．関数，手続き（関数定義と関数の呼び出しの区別）

これらの項目については，演習に時間を取るようにして定着を図ることが必要である．項目１～４までは第１学期の範囲，項目５は第２学期の範囲で行う．

言語によらない障壁として，ものづくりの基本が定着しないという問題がある．

６．ものづくりの基本１）　手がける順序
　①検査部分（出力）　②設定部分（入力）
　③処理部分
７．ものづくりの基本２）　部分完成と補完追加完成
　①少しずつ作る
　②幹から枝へ，低層から層を重ねるなど
　③正常系が完成してから異常系を手掛ける
８．ものづくりの基本３）　基本設計を重視
　①手戻りを少なくする工夫
９．ものづくりの基本４）　完成後の吟味・検証
　①吟味・検証のための基準や道具をつくる
10．ものづくりの基本５）　作成のための道具（治具）
　①種々のプログラミングユーティリティを使いこなす

これらについては，折りあるごとに演習を通じて実施を促しているが，全員への定着には至らない．

【終わりに】

ここに紹介したプログラミングの事例は，情報処理を専門とする学科の学生に対するものであり，情報処理を専門としない学科の場合には，また違った言語の選択と対応が必要である．そのような学科の学生は，既存のツールやライブラリを利用して本来の専門の修得に役立つ IT スキルを身に着けることがより重要である．

その場合，プログラミングが修得し易く，かつ PC の他のアプリケーションなどとの連携が容易な言語，例えば Python などが選択肢の一つとして考えられる．もちろん，学生の PC 環境や，インストールやメンテナンスの容易さなどの他の要素も考慮する必要がある．

文　献

１）経済産業省：IT 人材の最新動向と将来推計に関する調査結果.
http://www.meti.go.jp/press/2016/06/20160610002/20160610002.pdf（閲覧日2017年8月19日）