社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2009－CE－98（30）
2009／2／28

C言語によるプログラミング教育についての省察

阿部圭一†

C言語を用いたプログラミングの教育には，大別して，C言語を教えるという立場と，Cという言語を使ってプログラミングを教えるという立場がある．後者を目指した場合，C言語を用いるのはきわめて教えにくい．

本報告では，C言語によるプログラミング教育の経験から得たいくつかの視点，すなわち，（1）C言語の初心者向け部分集合であるBasic Cの提案，（2）プログラミングへの導入のしかた，（3）繰り返しの教え方，（4）プログラミングを教えることの本質的な難しさの原因などについて，提案と反省点を述べる．

Considerations to Teaching Programming in C Language

KEIICHI ABE†

There might be various purposes of teaching programming in C language. One extreme is teaching C language, that is, its grammar and usage, and the other extreme is teaching programming just using C language as a vehicle. In this report the latter is considered, and some proposals and reconsiderations are presented. They are (1) proposal of Basic C as a subset of the C language for novices, (2) how to introduce programming, (3) how to teach programming with iterations, and (4) the reason why programming is so difficult for some novices.

1．はじめに

手続き型言語を用いたプログラミング教育において，現在ではC言語あるいはC＋＋言語を用いている組織は多いと思われる．筆者の所属する愛知工業大学経営情報科学部情報科学科もその一つで，筆者はその初級の科目「プログラミング及び演習Ⅰ」と中級科目「プログラミング及び演習Ⅱ」とを，過去3年間担当した．その経験に基づいて，C言語によるプログラミング（以下，Cプログラミングと記す）についていくつかの反省と提言を述べる．なお，C＋＋を用いたプログラミング教育については筆者には経験がないので，C言語の範囲に留める．

筆者は，プログラミングの教育は，プログラミングの仕方を教えることであるべきで，特定のプログラミング言語を教えることではないという立場をとる．プログラミングを教えるには何らかのプログラミング言語によらざるを得ないが，選んだ言語にできるかぎり依存しないプログラミングの方法論を教えるのがプログラミング教育であると信じる．以下，この立場に立って，C言語によってプログラミングを教えようとするときの問題点と提案を述べる．

そもそも，CはD. M. Richieが，UNIXの開発のために考案した言語で，言わば「プロ中のプロ」の手によって「プロ向きの仕事用に」設計・開発された言語である[1]．これをプログラミングの初心者教育に用いようとするのには，本来無理がある．しかし，実用的に最も広く用いられているプログラミング言語の一つであるから，プログラミングを教えるための言語としてこれを選ぶことには筆者も同意せざるを得ない．

C言語の特徴は次のようであると言われている．

†愛知工業大学
Aichi Institute of Technology

（1） 言語仕様およびコンパイラがコンパクトである．
（2） プログラムの移植性に優れる．
（3） 簡潔なプログラムを書くことができる．
（4） ハードウェアの性能を高度に引き出すプログラムを書くことができる．
（5） 演算子が豊富である．

しかし，これらの特徴のうち（3）－（5）は，裏を返せば初心者の学習には必ずしも適切ではないことを意味する．

2．Basic C の提案

英語という言語にたいしては，C. K. Ogden により英語を母語としない学習者を対象として英語の語彙や文法を制限した Basic English が提案され，それに基づいた読み物も多数出版されている[2]．これに倣って，C言語にたいしてもそれを制限した部分集合 Basic C を考案し，初心者の C プログラミング教育の第一段階では Basic C の範囲で教えるという方策が考えられる．筆者が代表著者となった教科書[3]はこのような方針で執筆したが，明確に Basic C を提案してはいなかった．本報告では，Basic C の叩き台を表 1 に示す．

表 1 の各行にはC言語の各機能別を，列には Basic C に含める機能，強制する記法，制限する機能，制限しても含めてもよい機能を記した．ただし，Basic C に含める機能の欄には，含めて当然という機能は省略し，含めることを特記すべきだと考えた機能のみを記した．

筆者は，Basic C の範囲においても，授業における機能・概念の紹介は，逐次（step by step）方式で行う．例えば，入力・処理・出力については，まず出力だけのプログラム（hello world），次に処理と出力からなり，入力値は代入文で与えるプログラムの順に導入する．ついで，入力値をプログラム中に代入文で書くのではなく，同じプログラムで実行時に入力値を変えられるようにするために，scanf 文を導入する．

選択構造については，if 文と else 文をまず紹介し，select 文は反復構造を終えた後で導入する．

反復構造については，while 文を用いて基本的な繰り返しの学習（4．を参照）をした後，多くの繰り返しが for 文を用いるとコンパクトに書けることを示す．（for 文を先に学習する行き方もあると思う．）do-while 文は，繰り返しの本体を 1 回以上実行しなければならない場合にだけ用いるよう指導する．これに関連して，while 文と for 文では繰り返しの本体を 1 回も実行しないことがあること，繰り返しではそのような 0 回実行も許すほうが一般的であることも説明する．

繰り返しの途中から脱出する break 文は，これらの文に関する演習を十分行った後で導入する．

以上の逐次方式の反対の説明のしかたは一覧方式であろう．選択構造なら選択構造，反復構造なら反復構造のために用いられる文の種類を一度に紹介する，演算子もある範囲でまとめて示す方法である．私見では，初心者向けのCプログラミングの授業では一覧方式は適さないように思う．しかし，世の中に流布する多くのC言語あるいはCプログラミングに関する（しかも安価なので学生が飛びつきやすい）本には，一覧方式に寄ったものが多いように感じる．

逐次方式に関連して，近年，次のようなことも感じている．筆者は，前に勤めていた大学の時代から，講義 1 コマ＋演習 1 コマの，2 コマ続き 3 単位の授業では，最初の 1 コマで講義をし，次の 1 コマで今講義した内容に関するコンピュータ使用の演習を行っていた．しかし，このやり方だと，かなりの学生が講義を真剣に聴いていなくて，演習の時間になってからあわてて講義内容の勉強を始めるのである．

紙の上での小さな演習も含めて，もっと短い間隔で講義と演習とのフィードバックを行うべきだと感じる．しかし，それを行うと，講義の途中でたびたびほぼ全員の演習完了を待って同期をとる必要があり，時間効率が悪くなるのが痛しかゆしである．

3．プログラミングへの導入

3.1 「プログラムとは何か」の説明

プログラミングへの導入として，「プログラミングとは何か」，「プログラミングはなぜ必要か」を教えることは意外に難しい．

電卓を用いた計算と比較するのは一つの入り方であると考える．次のような説明をする．電卓で，例えば，ある時刻（時・分）からある時間（時・

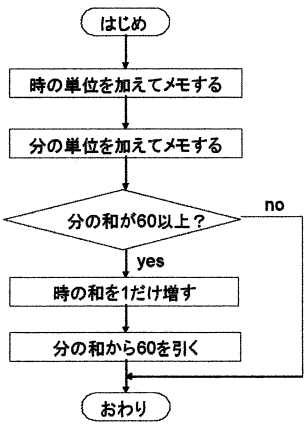

図 1　電卓による計算過程の例

表１　Basic C の提案

	Basic C に含める機能	強制する記法	制限する機能	制限しても，含めてもよい機能
scanf, printf と計算	注釈		int, double 以外の整数型，浮動小数点型，8進数，16 進数，long 定数	
選択構造	インデンテーションを含むスタイル	選択される実行文のブロックが1文でも{と}で囲む	条件式の中で，値が0ならば偽，それ以外は真となることを利用する書き方	&&と||は，最初は導入しない
反復構造	同上	同上 switch 文の各 case は必ず break 文で終わる	同上．ただし，永久ループを表す while(1)だけは認める．continue 文，goto 文，名札	do-while 文（含めるならば，繰り返しの本体を1回以上実行する場合のみに用いるよう指導する）
プリプロセッサ			#define は記号定数のみに使い，マクロは使用しない 自分で作成したヘッダ・ファイルの#include	
関数 変数のスコープ	引数の参照渡し，再帰呼び出し		記憶域クラス指定子 extern, register	広域変数， 記憶域クラス指定子 static
配列	多次元配列			
文字，文字列	文字コード， ¥n などの拡張表記	文字を扱うと，scanf は改行の扱いに問題があるので，fgets と sscanf を組み合わせた代わりの関数を提供する		
演算子	キャスト演算子， 演算子の優先順位		前置増分演算子（例：++i）， 前置減分演算子（例：--i）， 配列の添え字としての増分／減分演算子（例：a[i++]）， 条件演算子?:， 複合代入演算子（+=等）， 順次演算子，	ビット演算子，シフト演算子（ただし，C のデータが実際にはどのようなビット列で表されているかを確認させる演習を行うには必要）
ファイル			fread, fwrite, feof, ferror, perror	
コマンドライン引数				コマンドライン引数
ポインタ			ポインタ演算，配列のポインタによるアクセス， 関数ポインタ	malloc, free
型定義 列挙型 構造体 共用体		構造体は型定義して使う	共用体	
大規模プログラム			ファイル分割	

分）だけ経った時刻を求めるという計算のやり方を考えさせる[1]．解答の一つは図１のようになるであろう．人は，このような流れに沿って，次はどのような演算を行うのかを考えながら電卓での計算を進めていく．

しかし，この計算をコンピュータで実行する場合には，次にどのような演算を行うのかをその場その場で指示するのはきわめて効率が悪い．なぜなら，現在のコンピュータは，パソコンですら１秒間に数10億回の演算を行うことができるからである．もし電卓を使う場合と同じようにそのつど次の演算を指示するならば，コンピュータで数10億分の１秒実行し，人間が何秒かかけて次の演算を指示することの繰り返しになってしまう．これではちょっと大きな計算には大変な時間がかかってしまい，人間はその間ずっとついていなければならないことになる[2]．

そこで，コンピュータを用いた計算や情報処理では，実行する多数の演算を「最初に何をする，次には何をする，・・・」と，あらかじめ全てまとめて与えておく．先の電卓での計算の例でのように，ある条件が成り立つかどうかによって実行する演算群が異なるときには，両方の場合の実行手順を記述しておく．このように，コンピュータに次々に実行させる演算を全てあらかじめ記述したものが「プログラム」であると説明する．あらかじめ全ての場合にたいする実行手順を与えておくことによって，コンピュータは自動的に次々に演算を実行できることを理解させる．文献[4]では，この概念を「手順的な自動処理」と呼んでいる．なお．日常用語で用いられる「コンサートのプログラム」「運動会のプログラム」との類似性についても触れる．

3.2 「プログラミングとは何か」の説明

次いで，「プログラムを作成する」ことが「プログラミング」であると述べるわけである．ただし，プログラミングとは単に「プログラムを書くこと」ではなく，図２に示す過程（ソフトウェア製作過程＝広義のプログラミング）であると説明すべきである．

初心者向けのプログラミングの授業においては，この図の「コーディング」を中心として，「プログラム設計」と「テスト」のごく一部をカバーするだけであると説明する．初心者に図２のソフトウェア製作過程が実感として把握できるわけではないが，プログラミングが単にプログラムを書くという単純な作業ではないことは述べておくべきであると考える．ソフトウェア製作過程の全体像を垣間見せる方法として文献[5]の試みは注目に値する．

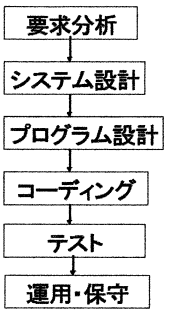

図２　ソフトウェア製作過程

この後に，図３に示すような，エディタによるプログラムの作成→コンパイル→実行という流れや，ソース・プログラム，オブジェクト・プログラムの概念を説明する．（初めの段階では，リンケージ・エディタの役割は話さないほうが良いだろう．）

3.3 人間側から見たプログラムの説明

コンピュータを用いて行う仕事は，通常，入力・処理・出力の組み合わせである．最初に示すプログラムとして，このうちのほとんど「出力」のみからなるプログラムが使われることが多い．通称「hello world」プログラム[6]，あるいはそれと類似のプログラムである．

このプログラム，あるいはそれ以降に例として学生に示すプログラムの説明において，あるとき次のことに気がついた．プログラムを先に提示してから，そのプログラムがどう実行されて，何を行うのかを後で説明するという順序をしばしばとっていたのである．hello world プログラムについて言えば，こういうＣプログラムを作ると，ディスプレイに hello world と表示できます，という説明である．これはおかしい．

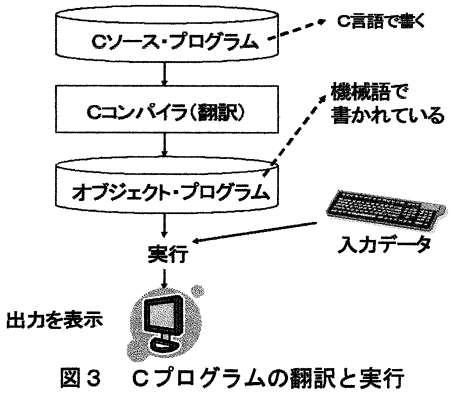

図３　Ｃプログラムの翻訳と実行

1)　この例題は蒲地輝尚：はじめて読むＣ言語，アスキー出版局，1991 によった．

2)　以前には，「コンピュータは高価なので，このような使い方は無駄が多い」ということも述べていたが，もはやこれは言及しないほうがよいかもしれない．

プログラミングは，「コンピュータを使ってやりたいこと」（問題）が初めにあり，それを行う（解決する）には，どのようなプログラムを書いて実行すればよいかを考える過程である．だから，ディスプレイに hello world と表示したい，それにはこういうCプログラムを書けばよいという説明の順序であるべきである．多少とも実用に近い規模のプログラムを作成する例題の説明では，まず問題の説明があり，次いでデータ構造の設計やモジュール設計の考察があり，最後にソース・プログラムが示される．つまり，図2のソフトウェア製作過程をある程度なぞる形で，説明がなされる．小規模のプログラムの例題においても，問題の提示→解決法→プログラムという順序にすべきであると反省した．この説明順序は「人間側からの説明」と言えるだろう．

しかし，Cプログラミングに関する本では「初めにプログラムありき」型の説明がまま見られるし，自分でも気づかないうちにそういう説明をしていることも多い．これは「コンピュータ側からの説明」と呼ぶべきであろう．そもそも「入力」「出力」という命名じたいが，コンピュータ側から見た名称になっている．これはもう変えようがないが，初心者向けの情報教育において，「プログラムが走る」「コンピュータが0を1つ余計にプリントした」など，人が主体でなくプログラムやコンピュータが主体であるかのような表現（擬人化）の濫用には，注意を払うべきだと考える．このような擬人化表現は，コンピュータおよびその上で実行されるソフトウェアをブラックボックスとして捉える傾向を助長していないであろうか？

4．繰り返しの教え方

反復構造（繰り返し）を含むプログラムを正しく作ることは，初心者の多くにとって難関の一つである．しかし，筆者はまず表2を示して，反復構造を持たないプログラムは，実用上無意味であることを説明する．「命令」を数える単位はあいまいであるが，その厳密な定義はここでは本質的でない．繰り返しを必要としない計算や情報処理は，プログラムを書いてコンパイルしてコンピュータで実行する（さらに誤りを修正して繰り返す）よりもずっと短い時間で，電卓で計算したり手作業で処理したりできるからである．

筆者は以前に，プログラムの反復構造（繰り返し）を考えるには，次の順序で考えるとよいと提案した[7]．

(1) 何を繰り返せばよいか．（繰り返しの本体）

表2　プログラム中の命令数と実行される命令数の比較

逐次構造	プログラム中の命令の個数＝実行される命令の個数
選択構造	プログラム中の命令の個数＞実行される命令の個数
反復構造	プログラム中の命令の個数＜実行される命令の個数

(2) 繰り返す部分は毎回同じ処理か．そうでないとしたら，何が変わっていくか．（更新処理[3]）

(3) どういう条件のあいだ繰り返すか．（繰り返すための条件）

(4) 繰り返しを始める前に行わなければならない処理は何か．（前処理）

(5) 繰り返しを終わった後で行わなければならない処理は何か．(後処理：これを必要としない繰り返しが普通である)

しかし，初心者の中にはこのような考え方でも壁を乗り越えられない者も居る．そこで，もう一工夫して，これを逐次方式で説明することを考えた．すなわち，次のように，繰り返しを構成する上の(1)-(4)の要素を1つずつ追加していく[4]．

(a) 単純にある処理を繰り返す．（当然これは永久ループになるので，プログラムとしては誤りである．）

(b) (a)に繰り返すための条件を付け加える．

(c) (b)に必要な前処理を付け加える．

(d) 繰り返す処理が系統的に変化する例を示して，(c)に更新処理を付け加える．

具体的には，次の例を用いた．ただし，これにはまだ満足していない．

(a) 整数を1個読み込んで，その値と，その2乗を表示することを繰り返すプログラムを，繰り返しの条件なしで提示する．

```
#include <stdio.h>
int main( void )
{
    int v;
    while( 条件 )  {
        scanf( "%d", &v );
        printf( "v = %d    v*v = %d¥n",
                v, v*v );
    }
    return 0;
}
```

3) 文献[7]では「増分」と記載していたが，「更新処理」のほうが学生にわかりやすいと考えて変更した．

4) A. W. Biermann: Great Ideas in Computer Science, The MIT Press, 1990，和田栄一訳：やさしいコンピュータ科学，アスキー出版局，1993の第3章からヒントを得た．

(b) 読み込まれる値は正の整数に限るものとして，0または負の整数が読み込まれたらやめるものとする．

```
#include <stdio.h>
int main( void )
{
   int v;
   while( v > 0 )  {
      scanf( "%d", &v );
      printf( "v = %d    v*v = %d¥n", v,
              v*v );
   }
return 0;
}
```

このプログラムは正しくない．最初に while 文の条件判定に来たとき，v の値は未定であることを指摘し，v の値の初期化を追加した(c)のプログラムを示す．

(c)

```
#include <stdio.h>
   int main( void )
   {
      int v;
      v = 1;
      while( v > 0 )  {
         scanf( "%d", &v );
         printf( "v = %d    v*v = %d¥n", v,
                 v*v );
      }
     return 0;
   }
```

(d) 1から10までを，各行に値一つずつ表示するプログラム

```
#include <stdio.h>
int main( void )
{
   int count;
   count = 1;
   while( count <= 10 )  {
      printf( "%d¥n", count );
      count = count + 1;
   }
return 0;
}
```

5．プログラミングを教えることの困難さ

綾皓二郎の言うように[8]，一般の人にとってプログラムを書くこと，それも与えられた問題にたいする正しいプログラムを書くことは，意外に難しい作業なのである．筆者の長年の経験からの結論としては，プログラミングには適性があると結論せざるを得ない．得意な者と不得意な者とが居る．プログラミングを教える教師は，たぶんプログラミングに適性のある人であったろう．それゆえ，自分にとってプログラミングの学習のスタートが容易であったものだから，プログラミングは本来難しいのだとか，不得意な者が居るとかいう事実を過小評価しがちである．

5．1　プログラムのパターンとその組み合わせ

付録に例を示すように，プログラミングにはいくつかの定型的なパターンがある．初心者レベルのプログラミングとは，与えられた問題にたいして，これらの中から必要なパターンを選んで（多少変形して）組み合わせ，足りない部分は自分で考えて補うプロセスであると考える．

パターンの組み合わせ方には，図4(a)，(b)に示す並置型の組み合わせと，(c)に示す入れ子型の組み合わせとがある．一般に，(c)の入れ子型の組み合わせは，(a)，(b)の並置型に比べて学生に理解されにくい．また，入れ子型の組み合わせはプログラムを作成するときに思いつきにくいようである．しかし，実際のプログラミングは，並置型と入れ子型の組み合わせを何重にも用いて，図4(d)に簡単な例を示すような構造を作っていく作業である．

初歩的な例として，選択構造（場合分け）と反復構造（繰り返し）を学習した後に，次のような課題を与えるとする．

［課題1］元号を表す1文字 M（明治），T（大正），S（昭和），H（平成）のどれかと，その元号での年を読み込み，西暦に変換して表示することを繰り返すプログラムを作成しなさい．上記の4文字以外の文字が入力されたら，実行を終了するものとする．ただし，明治元年は1868年，大正元年は1912年，昭和元年は1926年，平成元年は1989年である．

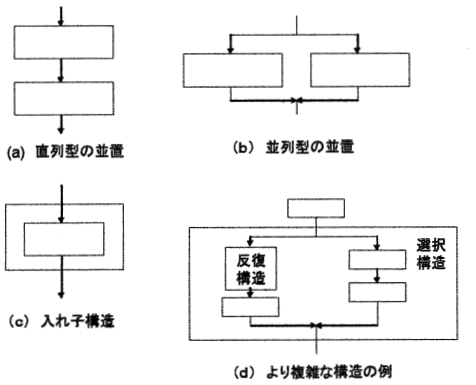

図4　プログラムのパターンの組み合わせ方

この解答となるプログラムが，繰り返しの中に場合分け（switch 文あるいは if-else の鎖）を含む入れ子構造になることを全く思いつかない学生が居る．

この場合のヒントの与え方としては，２通り考えられる．

（１）解答となるプログラムの外側の構造は何であるかを考えさせる．つまり，それがなにかを繰り返す構造であることを発見させる．その後に，何を繰り返しせばよいかを考えさせ，それが選択構造であることを発見させる．

（２）とは逆に，内側の処理から考えさせる．このためには，上記の課題１の前に次の課題０を示して考えさせるとよいだろう．

［課題０］元号を表す１文字 M，T，S，H のどれかと，その元号での年を読み込み，西暦に変換して表示するプログラムを作成しなさい．M, T, S, H 以外の文字が入力されたときは，「誤りです」と表示しなさい．

この課題にたいするプログラムを作らせた後に，その選択構造を繰り返して課題１のプログラムに到達させるようにする．

筆者は，構造的プログラミングの立場から，従来はトップダウンに考える（１）の考え方を奨励してきたが，step by step の立場からは（２）のほうが学生には考えやすいようである．

一般に，このような組み合わせ型の思考を必要とする問題にたいしては，学生の力はしだいに衰えてきているような気がする．多くの学生の考える「勉強」とは，極論すれば「例題 A の答が a なら，例題A’の答は a ’であろう」的な，一問一答方式あるいは個別撃破方式になってきているのではないか？

これにたいして，筆者はプログラミングの演習課題として，先週やった○○と，今週やった△△と，前学期に習った□□を組み合わせて考えると目的のプログラムが作れるといった問題を出しがちである．このような組み合わせを考えることこそが，プログラミングの本質であると筆者は主張したい．また，このような組み合わせ的な（総合的な）思考を要求する訓練は，受験勉強において横行している一問一答方式あるいは個別撃破方式に染まった思考法を揺さぶるという効果があるはずである[4]．現実社会では，一問一答式のクイズのチャンピオンなどほとんど役に立たないからである．

そもそも，近代科学の主要なパラダイムは分析（analysis），すなわち divide and conquer であった．学校教育においても，理系科目だけでなく文系科目においてすらこのようなアプローチが主流として教育が行われている．それにたいして，プログラミングは明らかに合成（synthesis）型の問題である．

しかし現実には，学生によってはこのような組み合わせ思考が苦手な者も居る．それに適応するための教師側の簡単な逃げ道は，「プログラミング」という看板を掲げながら，プログラミングを教えるのではなく，C言語の文法を教えることである．表１の左端の列に示すような単元別に例題を用意し，それをちょっと変形した演習課題やテスト問題を出せば，より多くの学生がプログラミングの授業に合格できるだろう．だが，それは筆者にとっては，C言語の文法を教えることであって，（言語に依存しない）プログラミングの方法を教えることではない．

もちろん，上記の議論は極端な二分法をとっての論であって，実際にはその中間的な教え方のさまざまなバリエーションが存在する．したがって，C言語の文法に重点をおいた教え方が，すべてプログラミングの教育に役立たないと主張するものではない．ただ，筆者はそういう行き方をとらないだけである．また，この議論にたいする立場は，次節に述べる「何のためにプログラミングを教えるのか」によって違うであろう．

５．２　何のためにプログラミングを教えるのか

最近，筆者はプログラミングの授業の目的がわからなくなっている．理工系情報専門学科の卒業生でも，実用に広く使われるプログラム，すなわち F. P. Brooks, Jr.の言う「プログラム・システム製品」[9]を作成する基礎的能力を持った学生は半数よりも少ないと筆者は推定している．情報専門学科以外の理系学科では，卒業研究や大学院での研究において，その目的限りのプログラムを書く機会を持つ学生は多いであろう．では，文系の学部・学科や一般教育においてプログラミングを教える目的や意義は何であろうか？

情報処理学会情報処理教育委員会の提言[4]では，大学の一般教育において，「手順的な自動処理」についての制作体験をさせることを提案している[5]．「手順的な自動処理」の内容は必ずしもプログラミングに限定しておらず，問題解決の手順をアルゴリズムとして記述し，それに従ってコンピュータで自動処理させる過程一般に広げている．その目的は，「手順的な自動処理」の理解にあると思われる（詳細は，文献内に記された「自動的な手順処理」の定義，およびそれを初等中等教育における教育実践手段とすることが有効な理由の説明を参照）．

では何のために「手順的な自動処理」の実践的理解が文系の学生にとっても必要なのであろう

---

5) この点を思い出させてくださった桃山学院大学の藤間真先生に感謝する．

か？　上記の提言には，「このような体験的理解をもった人であれば，学んだことがなく全く新たに遭遇するような場面においても，体験的理解を土台として「コンピュータはこのように動作するだろう」「したがってこういうことはできるが，こういうことはできないだろう」という判断をある程度的確に行えるはずである」と記されている．筆者もこれに賛成であるが，この周辺をもう少し深く考えてみる必要があると感じている．

５．３ プログラミング以前の学び

プログラミング以前に学習しておくべきこととしては，従来，コンピュータに関する基礎知識が挙げられていた．例えば，コンピュータを構成する機器，コンピュータの動作のしくみ，命令とデータの区別，２進法，データの表現法，オペレーティング・システムの役割などである．

ここでは，上記のほかに，プログラミング以前に，論理的な考え方とそれを表現する方法の訓練を行うことが効果的ではないかという仮説を提示する．

論理的な考え方とそれを表現する方法の訓練として何が有効であるかについては，経験が蓄積されていないように思われる．離散数学の演習，論理パズル，論理的な文章を書かせることなどは有効ではないかと考えているが，実証できる根拠は持っていない．論理的な文章を書かせるための課題として，李元揆は知恵の輪を外す過程を，筆者は論理パズルを解く過程を書かせているが，適切な題材の範囲をもっと増やす必要がある．

６．おわりに

Ｃ言語によるプログラミング教育の経験に基づいて，いくつかの反省点の披露と提案を行った．教育においては，経験により得た知見を他人に伝わる客観的情報として提示し，共有できるようにすることが重要であると考える．そういう観点からの報告である．

謝辞

特定のお名前は出さないが，プログラミングを含む情報処理教育に関して多数のご教示・ご助言を賜った多くの方々に感謝する．

参考文献

1) B. W. Kernighan and D. M. Ritchie: The C Programming Language, Second Edition, Prentice Hall Software, 1988，石田晴彦訳：プログラミング言語Ｃ 第２版，共立出版，1989
2) Wikipedia: Basic English http://en.wikipedia.org/wiki/Basic_English
3) 阿部圭一他：プログラミング，オーム社，1999
4) 情報処理学会情報処理教育委員会：日本の情報教育・情報処理教育に関する提言 2005（2006.11 改訂／追補版），http://www.ipsj.or.jp/12kyoiku/teigen/v81teigen-rev1a.html，2006
5) 荒木恵，松澤芳昭，杉浦学，大岩元：プログラミング教育への導入のための情報システム概念に基づくアンプラグドワークショップ，SSS2008 情報教育シンポジウム報告集，pp.163-170，2008
6) 1)の p.8
7) 阿部圭一：ソフトウェア入門　第２版，p.34，共立出版，1983
8) 綾皓二郎：高校の次期情報教育カリキュラムにおける計算機科学・計算機工学教育を検討する，2007PC Conference 論文集，pp.127-130，CIEC（コンピュータ利用教育協議会），2007
9) F. P. Brooks, Jr.: The Mythcal Man-Month – Essays on Software Engineering, Addison-Wesley, 1975, 1996，滝沢徹，牧野祐子，富澤昇訳：人月の神話：狼人間を撃つ銀の弾はない，アジソン・ウェスレイ・パブリッシャーズ・ジャパン，1996

付録　プログラムのパターンの例

（１）if-else の鎖

```
if( ＜条件１＞ ) { ＜文１１＞; ＜文１２＞; ・・・ }
else if( ＜条件２＞ ) { ＜文２１＞; ＜文２２＞; ・・・ }
else if( ＜条件３＞ ) { ＜文３１＞; ＜文３２＞; ・・・ }
     :
else  { ＜文ｎ１＞; ＜文ｎ２＞; ・・・ }
```

（２）繰り返しの基本形

```
＜初期化＞;
while( ＜条件＞ ) {
    ＜本体＞;
    ＜更新処理＞;
}
＜後処理＞;          /* もし，あれば */
     （＜更新処理＞が＜本体＞より前に来る場合もある．）
```

（３）何かを数える

```
count = 0;
ここから繰り返しに入る  {
     :
   数えるものが現れたときに
       count = count + 1;
   :
}
/* count に個数(人数，・・・）が入っている */
```