プログラムを読むことによるアルゴリズムとプログラミング教育

関口久美子

1．はじめに

コンピュータで動作するシステムを開発するということは，いくつかの設計工程を経て，最終的にはプログラムを作成することであり，言い換えればプログラムを書くということである。そのため情報系の教育では，当然「プログラムを作成できること」を目標としてアルゴリズム教育やプログラミング教育が行われている。一方実際の業務では，まったく新規にプログラムを作成するよりも，むしろ既存のシステムへの機能の追加や変更，現システムの誤りの修正，あるいは既存プログラムの再利用などのほうが多いのが現状である。さらにはソフトウェアの扱いにおける時代の変化もある。プログラムのソースコードは極秘とし，供与するときはライセンス料を取るという時代から，ソースコードを無償で公開し，だれでもそのソフトウェアの改良，再配布が行えるというOSS（オープンソース，またはオープンソースソフトウェア）と言われる開発環境の時代に変化してきている。

このような現状で必要とされるのは，他者が作成した既存のプログラムを正しく読み取り，それに対する変更箇所と内容に見当をつけられる力である。最近では「CodeReading」などを皮切りに，プログラムの読み方を扱った書籍[1) 2)]が多数出版されている。また，情報系の資格試験として認知度の高い情報処理技術者試験（経済産業省認定の国家資格）などにおいても，プログラムを読み取る力なくしては合格することは難しいことからも，読み取る力の必要性は明らかである。これらは単に時代に応じた要請だけではなく，コンピュータが一部の人のものであった時代から，「プログラムの読み方」[3)]として提唱されていたのである。

プログラムが読めるとは，プログラムリストからそのプログラムにおいてどのような処理が行われ，それによりデータ領域の内容がどのように変化し，どのような結果が得られるかを推測できることである。プログラム

が読めることと書けることは別ものではない。プログラミングという作業は，プログラムを書き，その誤りを捜し，さらに正しく直すことの繰り返しであり，そのためにはプログラムを正確に読める必要があるのである。読めなければ正しいプログラムは書けないとも言える。

しかしながらアルゴリズム教育や学生の資格取得の支援を行なっている立場から，このことに危機感を感じずにはいられないのが事実である。問題解決能力の低下，基礎学力の低下，大学全入時代の弊害等々，その原因は様々なところにあると言えるが，その状況に応じた教育方法を行うことが必要である。そこで，本学の学生における「プログラムを読み取る力」の現状を調査することにより，アルゴリズム教育とプログラミング教育をとおしてプログラムを読み取る力を向上させる学習方法を探ってみることにした。

2．調査

埼玉工業大学工学部情報システム学科の「アルゴリズムとデータ構造Ⅱ」の講義受講者（2年～4年）を対象に調査を行った。延べ51名の回答があったが，25の調査項目を2回に分けて実施したため，2回とも回答した学生40名分のデータのみを結果として扱った。

調査は，あるアルゴリズムを学生が学習済みである流れ図とC言語のプログラムにより表記したものをそれぞれ下レース（流れ図やプログラムの命令を机上で一つひとつ追う作業）をしてもらい，正しいデータ内容が得られたか否か，その正答率を調査することで行った。用いたアルゴリズムは，(1)1次元配列のデータ構造を用いた繰り返し処理が正しく読み取れるか，(2)2次元配列のデータ構造を用いた2重の繰り返し処理が正しく読み取れるか，(3)プログラム中で用いるデータ名を長くした場合，読み取りにくさに影響があるかを調査することを目的とした内容である。また，C言語のプログラムにおいては四則演算やデータ型，あるいは画面への出力，キーボードからの入力処理，分岐命令などの基本命令が正しく読み取れているかを確認した。プログラムで用いるデータ名は，本来その内容を推測できるようなネーミングをするのが一般的であるが，ここではトレースの正確性を調査するため，名前からは推測できないような名称を用いた。

3．結果

調査の結果を表1に示す。

表1　調査項目の内容と正答率

	項番	調査項目	詳細	正答率(%)
C言語プログラム（基本命令）	1	四則演算	加算（＋演算子）	100
	2		減算（－演算子）	100
	3		乗算（＊演算子）	95
	4		除算（／演算子）	78
	5		除算の余り（％演算子）	70
	6	データ型	小数点を含む値の実数型(float型)への代入	89
	7		小数点を含む値の整数型(int型)への代入	64
	8	入出力	printf命令による画面への表示	99
	9		scanf命令によるキーボードからの入力	100
	10		printf命令とscanf命令の組み合わせ	91
	11	分岐処理	if ～ then ～ else 型	80
	12		if ～ then ～ elseif 型	93
C言語（配列）	13	1次元配列(1)	for命令による繰り返し（要素の合計）	65
	14		for命令による繰り返し（要素の移動）	50
	15		配列を図示する	45
	16	1次元配列(2)	長いデータ名・13との比較	58
	17		長いデータ名・14との比較	45
	18		配列を図示する・15との比較	38
	19	2次元配列(1)	for命令による繰り返し（2重ループ）	13
	20		2次元配列を図示する	43
流れ図	21	1次元配列(3)	ループ端記号による繰り返し・13との比較	85
	22		ループ端記号による繰り返し・14との比較	65
	23		配列を図示する・15との比較	75
	24	2次元配列(2)	2重のループ端記号・19との比較	50
	25		2次元配列を図示する・20との比較	75

3－1　C言語プログラムの基本命令

項番1～12におけるC言語プログラムの基本命令は初歩的な問題であり，当然のことながら高い正答率を示している。

項番3の乗算において100％に至らなかったのは単純な計算ミスと考えられる。また，項番4の除算では演算子の意味は理解していたものの，整数どうしの除算の結果が小数点を含んでおり，その代入においてのデータ型に関する誤りであり，検査項目としては項番7に分類される内容であった。項番5の除算の余りは3割の学生が不正解であったが，普段使い慣れない演算子であったため，定着していなかったと考えられる。

項番6，7の項目は値をデータに代入するという単純な命令であるが，項番7の正答率は64％であり，このことからデータ代入時にその型のタイプを意識していない学生が多いことがわかる。

項番8～12の項目については，いずれも普段から使い慣れた命令であり，ほとんどの学生が理解し使いこなしていると言える。

3－2　1次元配列を用いた繰り返し処理

「配列」とは同じ型（属性）のデータを連続した領域に並べたデータ形式であり，プログラミングにおいて使用されるもっとも基本的なデータ構造である。左から何番目という添字（インデックス）を表名に添付し，それを変数として扱うことにより，少ない命令で多くのデータを処理することができる。配列は繰り返し処理と合わせて使用することが多いが，プログラミング学習初心者にとってはこの配列の扱いがプログラミング習得の壁になることが多い。本調査においても，項番13以降の調査項目の正答率が項番1～12までの基本命令に比べ極端に低くなっている。

項番13～15，21～23はいずれも同じアルゴリズムであり，流れ図の表記（図1）

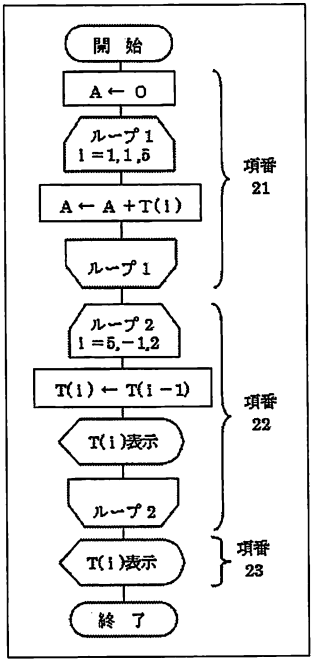

図1 流れ図による1次元配列処理

でトレースしたものと，C言語のプログラム（図2）をトレースしたものの結果である。

```
#include <stdio.h>

int main(void)
{
  int i;
  int t[5] = {12,5,23,4,11};
  int s = 0;

  for(i=0; i<5; i++) {
    s += t[i];
  }

  for(i=4; i>0; i--) {
    t[i] = t[i-1];
    printf("t[%d]:%d\n",
                    i,t[i]);
  }

  printf("t[%d]:%d\n",i,t[i]);
  return(0);
}
```

図2　C言語による1次元配列処理

項番13と21は配列Tの1番目から5番目までの要素を順にA（プログラムではs）に合計するアルゴリズムであり，ループ終了時のデータAの内容を答えさせた。繰り返し処理により，対象とする要素番号を1から5まで変化させる単純なものである。また，項番14と22は同じ配列を扱った繰り返し処理であるが，対象とする要素番号を5から順に1ずつ減じながら，要素の内容をひとつ後ろへ移動するものである。“T(i)表示”により出力される配列Tの内容を繰り返しのつど答えさせた。項番15と23では，終了直前での配列Tの様子をイメージする図で記述させた。

流れ図とC言語プログラムの2通りの表現方法による違いを比較するために，表1の項番13～15，21～23を表2にまとめ直した。

表2より，項番2-1に対し項番2-2では，流れ図およびC言語プログラムのいずれも正答率は85％から65％，65％から50％へと下がっており，その低下の割合は0.76であった。項番2-1は1番目から単純に合計をするという，常時使用する馴染みのある処理であったのに対して，項番2-2は配列の後方からの処理であり，しかも扱う配列が1行中に2つ出現しているためトレースする際の手間が増え，正答率が下がったものと思われる。

一方，流れ図とC言語プログラムでの状況を比較してみると，項番2-1では85％から65％へ，項番2-2では65％から50％へといずれもC言語の方が0.76，0.77の割合で下がっている。

項番2-3では，配列の内容が正しいか否かに関わらず1次元配列のイメージを図示できているものを正答とした。正答できなかった学生のほとんど

は何の回答もせず空欄であった。トレースする際には，プログラム内で使用している配列やデータ等を図示し，その内容がどのように変化するかを実際に書きこみながら流れを追っていく方法が読み取りやすいはずであるが，自らトレースをするための環境を作り出せないようである。

表2　1次元配列を用いた繰り返し処理

項番	調査項目	正答率（%）		割合 C/流れ図
		流れ図	C言語	
2-1	配列要素の合計処理	85	65	0.76
2-2	配列要素の後方への移動処理	65	50	0.77
2-3	配列を図示する	75	45	0.60

3－3　2次元配列を用いた繰り返し処理

2次元配列の処理としてパスカルの三角形を作成するアルゴリズムを用いた。パスカルの三角形とは，$(a+b)^n$の2項展開における係数を三角形状に並べたものであるが，ここでは三角形でなく2次元配列に左寄せした形状で作成している（図3）。このアルゴリズムを自分で考えて作成することはかなり高い知識と技術を要求されるが，ここでは単に2重の繰り返し処理と2次元配列への操作を正確に読み取るということだけが課題になっている。流れ図の表記とC言語のプログラムを図4と図5に示す。

1				
1	1			
1	2	1		
1	3	3	1	
1	4	6	4	1

図3　2次元配列上のパスカルの三角形

表3はこの2通りの表現方法による違いを比較するために，表1の項番19～20，24～25をまとめ直したものである。

項番3-1では，5行5列の2次元配列にパスカルの三角形の係数が正しく作成されていることを確認した。しかしその正答率は，流れ図で50%，C言語では13%と極めて低い値となった。ここでの回答では2次元配列の行と列を逆に扱っている学生が数名いたが，トレースの正確さを調査するという観点からこの場合も正答として処理した。

項番3-2は，1次元配列の項番3-3と同様に，2次元配列のイメージを図示させた。ここでも，単に2次元配列の図が描かれていればデータの内容にかかわらず正答とした。

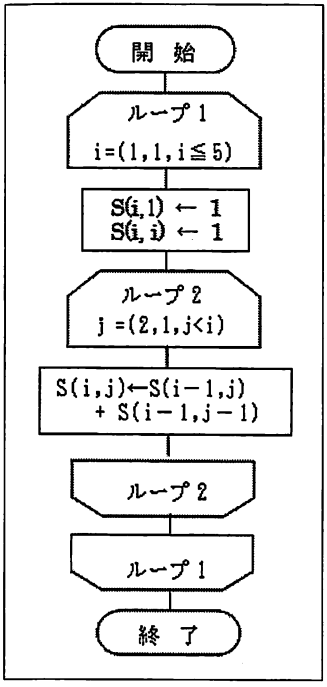

図4　流れ図による
2次元配列処理

```
#include <stdio.h>

int main(void)
{
  int i, j;
  int t_binomial[5][5] = {0};

  for(i=0; i<5; i++) {
    t_binomial[i][0] = 1;
    t_binomial[i][i] = 1;

    for(j=1; j<i; j++) {
      t_binomial[i][j]
        = t_binomial[i-1][j]
        + t_binomial[i-1][j-1];
    }
  }

  return(0);
}
```

図5　C言語による2次元配列処理

表3　2次元配列を用いた繰り返し処理

項番	調査項目	正答率（%）		割合 C/流れ図
		流れ図	C言語	
3-1	2重の繰り返し	50	13	0.26
3-2	2次元配列を図示する	75	43	0.57

3－4　長いデータ名を用いた繰り返し処理

同じアルゴリズムを同じプログラム言語に置き換えた場合でも，そのプログラムの書き方により読みやすさは様々である。特に初心者にとっては長いデータ名をつけることによりプログラムリストの字数が増え，見た目

の難易度が変わるのではないかという予想した。表1の項番16～18は，項番13～15で使用したC言語のプログラムを単に長いデータ名に変えたプログラム（図6）でトレースを行ったものである。

この2つのプログラムでの違いを比較するために，表1の項番13～15，16～18を表4にまとめ直した。項番4-1における短いデータ名と長いデータ名での比較では，65％から58％へ，項番4-2では50％から45％へと正答率は下がっているものの，その低下率はそれぞれ0.89と0.90であり，予想したほどの差はなかった。図6のプログラムのトレースが正答となった回答の中には，リスト上の長い名前を丸で囲み見やすくしたものや，あるいは別の短い名前に書き換えていたものがあった。読み方のコツやルールがつかめてしまえば，一見して難しそうに見えても，工夫によって読み解けることがわかった。

```
#include <stdio.h>
void main()
{
  int array_idx;
  int array_calc[5] = {15,6,7,21,3};
  int answer_calc = 0;

  for(array_idx = 0; array_idx < 5; array_idx++) {
    answer_calc = answer_calc + array_calc[array_idx];
  }

  for(array_idx = 4; array_idx > 0; array_idx--) {
    array_calc[array_idx] = array_calc[array_idx - 1];
    printf("array_calc[%d]:%d",
            array_idx,array_calc[array_idx]);
  }

  printf("array_calc[%d]:%d",
          array_idx,array_calc[array_idx]);
  printf("answer_calc:%d",answer_calc);
  return(0);
}
```

図6　長いデータ名によるC言語プログラム

表4　データ名の違いによる読みやすさ

項番	調査項目	正答率（%）		割合 短／長
		短い データ名	長い データ名	
4-1	配列の要素の合計処理	65	58	0.89
4-2	配列の要素の後方への移動処理	50	45	0.90
4-3	配列の内容を図示する	45	38	0.84

4．トレース指導の必要性

プログラムのトレースができない要因として，プログラム言語の特性がある。たとえばC言語は一見すると単なる英字や数字の羅列である。それを読むためには，(1)最低限必要なデータの宣言部と命令部を見つけ出すこと，(2)プログラムの大まかな制御構造をつかむこと，(3)プログラムの命令がどのような順序で実行されるかを知ること，(4)命令の実行によりデータ領域の内容がどのように変化するかを知ることなどが必要であるが，それらを順序立てて処理することが難しい。さらにプログラム言語には非常に多くのルールや制約があり，それを消化しきれない状況にある。また，配列のイメージ図を書けなかったことから明らかなように，命令の実行をプログラムが実行されるメモリ空間に対応付けられないこと，さらにはプログラムの実行そのもののイメージがつかめないことも大きな要因である。

表2の項番2-1および2-2において，流れ図とC言語のプログラムでは流れ図のトレースのほうがその正答率が高かった。流れ図は図であるためイメージとして捉えやすかったこと，また流れ図には命令以外の余分な定義が記述されていないため，C言語プログラムで必要な仕分け作業が必要ないことなどが読みやすさにつながっていると思われる。しかし，ただ単に見やすさだけが読めた要因ではない。アルゴリズム教育の講義にて流れ図をトレースするという作業を常に習慣づけてきた効果が表れていると考えられる。当該講義では，簡単なアルゴリズムを対象とした初期の段階からトレースの作業を行なってきた。特に配列を用いたアルゴリズムでは，必ずその配列の内容を図示し，一つひとつ処理を追っていくような指導をしてきたのである。しかし，アルゴリズム教育での教授法にも問題がある。黒

板や書画カメラで実際にやって見せるだけでは指導しきれず，個別の対応が必要になることがしばしばであり，大人数教育では手が行き届かない状況になっている。

これらから，プログラムの読み方は教えなければ読めるようにならないということが言える。もちろんそれぞれの表記法を少し教えただけで簡単に読めるようになる学生もいるが，そういった学生はごくわずかである。多くはトレースを行う環境を自ら整えることができないし，流れ図でのトレースの技法をプログラムのトレースとして応用することができないのである。

5．おわりに

プログラムを読む力を向上させるためには，アルゴリズム教育とプログラミング教育を効果的につなげることが不可欠である。本来であればアルゴリズム教育はプログラム言語に依存しないものであるが，異なった表記法での学習を互いに関連付けられない状況では，アルゴリズム教育での表現方法をプログラミング教育での表現に近づける必要がある。すでにアルゴリズム教育とプログラミング教育をあえて分離せずに行う教育[4]も報告されている。

また，プログラミング初心者にトレースを指導するシステム[5]も報告されているが，さらにプログラムの実行がイメージできるような学習支援システムの構築も必要である。それは初心者にとって英数字の羅列にしか見えないプログラムを熟練者が読み解くように仕分けし，さらにプログラムの文法がわからなくても命令の実行の様子をイメージできるように表示してくれる支援システムである。その利用によりアルゴリズムやプログラミング学習の効果が向上するだけでなく，コンピュータの仕組みについての理解も深まることが期待できる。

今後はこれらのシステムの構築および実践により，アルゴリズムとプログラミング教育の効果の向上を検証していきたい。

参考文献

1) DiomidisSpinellis/トップスタジオ訳（2004），CodeReading，毎日コミュニケーションズ出版
2) 高木信尚（2010）プログラマーのためのソースコードを読む技術，技術評論社
3) 森口繁一，小林光夫，武市正人（1981），プログラムの読み方，岩波書店
4) 新開純子，宮地功（2009）プログラミング学習支援システムを用いた入門教育の実践，日本教育工学会論文誌33(Suppl.),5-8
5) 江木鶴子，竹内章（2009）プログラミング初心者にトレースを指導するデバッグ支援システムの開発と評価，日本教育工学会論文誌，32(4):369-381