初心者にとってのアルゴリズム作り

林 雄二

要 旨

道情報大では，１学年において，プログラミング言語教育と並行して「プログラム設計論」を開講している．ここでは，「プログラム設計論」の目的とその内容にふれ，さらに，初心者がアルゴリズムを学ぶ段階でつまずく原因のいくつかを，今までの経験をもとにして列挙した．なかでも，初心者にとって発想が容易ではないと考えられる３つの点，すなわち，アルゴリズムの動的性，プログラムにおける変数の生かし方，アルゴリズムにおけるデータ操作と制御の混在，を中心に解説している．

１．はじめに

道情報大（経営情報学部）では，情報学科，経営学科の両学科１年生を対象に，半期で「プログラム設計論」を開講している．並行して行われている「電子計算機実習」の方でプログラミング（平成４年までは FORTRAN）を教育しているので，「プログラム設計論」はプログラミングを習い始めたばかりの学生が対象である．次章ではこの科目の目的と内容を紹介する．

「プログラム設計論」の教育を通して，アルゴリズム作りの考え方を身につけることが一朝一夕になるものではないことを，筆者はあらためて知らされた．中学，高校と数学を学んできていながら，数学のように特別な基礎知識を必要としない，いわば数学よりはるかに易しいであろうと思われるアルゴリズム作りに，なぜ学生は困難を感ずるのであろうか．数学の課題を考えることとどのように異なるのか，第３章ではその原因と思われる６つの点を列挙した．その中でも，アルゴリズムの動的性，プログラムにおける変数の生かし方，アルゴリズムにおけるデータ操作と制御の混在，の３点が初心者にとって重要な鍵であると考えられるので，例を含めて解説している．

Yuji HAYASHI 北海道情報大学経営情報学部

２．プログラム設計論

「プログラム設計論」の主な講義（できるだけ演習をまじえている）内容は以下の通りである．

(1)プログラム設計の位置付けとアルゴリズム
(2)フローチャートによるアルゴリズム表現
(3)構造化プログラミング
(4)疑似言語によるアルゴリズム表現
(5)ＰＡＤによるアルゴリズム表現
(6)ジャクソン法（ＪＳＰ）

簡単にこれらの内容にふれておく．(1)プログラム設計は工場における一般の製品開発と

同様に，重要な設計段階であり，その中心になるのがアルゴリズム作りであることを学ぶ．(2)プログラミング言語を修得していることを前提としないでアルゴリズムを学ぶために，その基礎的な表現手段としてのフローチャートを身につける．機械語がプログラミングの基盤であるのと同様に，フローチャートは必ずしも良い道具とは云えないにしても，アルゴリズム表現の基盤であると考えられる．(3)アルゴリズムが複雑化していく場合の問題点を考えながら，構造化プログラミングの考え方を学ぶ．プログラミング段階ではなく，プログラム設計段階における構造化プログラミングの中心は，基本3構造によるアルゴリズム作りである．(4)(5)構造化プログラミングを指向したアルゴリズムの表現法としてフローチャートは望ましいものではないので，疑似言語，ＰＡＤを学び，これらによってアルゴリズムを表現する訓練を行う．疑似言語はプログラミング言語の核の部分を含んだ文章的記法であり，一方図式的な記法としてはＰＡＤが優れている．(6)さらに，構造化プログラミングの方法論として，ジャクソン法（ＪＳＰ）も学ぶ．入出力データ構造を基本にしてプログラム構造を作れ，というＪＳＰは，けっして高度な知識，能力を必要とする方法ではないので初期アルゴリズム教育の一環として学ぶのが適当であろうと考えている．しかし残念ながら，ここ2年間は時間数（半期1コマ／週）が充分でないために，実施していないのが現状である．

良いアルゴリズムをいかにして作り，いかにして表現するか，これが本科目の目標である．プログラミング教育と切り放してアルゴリズム作りを学ぶのは，プログラミング言語の細かな約束ごとや操作コマンドなどを意識せずにアルゴリズム作りに考えを集中することが出来ると考えてのことである．初心者の段階から構造化プログラミングのような内容を果たすのは，構造が表面に表れるような分かり易いアルゴリズム作りを，できるだけ早い段階で身につけておくべきであると考えるからである．

3．初心者のつまづき

初めての人にとって，アルゴリズムを作ることはけっして容易な仕事ではない．誰しもが初心者の段階でつまづいた経験を持っているはずである．初心者がアルゴリズムを学ぶ上で障害になっているのは何か，その原因と思われるものを(1)～(6)として列挙してみる．

(1) アルゴリズムは，その実行に動的な変化を伴うものである．： プログラムの実行とは，1ステップづつ記憶内容(すなわち状態)を変化させながら処理を進めていくことである．紙の上に記述されたプログラムやチャートが意味しているのは，動的な状態の変化である．ダイクストラ（E.W. Dijkstra）も，構造化プログラミングのきっかけとなった彼の論文[1]の中でつぎのように述べている．「人間の知的な能力は静的な関係を把握するのにはうまくかみあうが，時間と共に変化するプロセスを認識する能力は劣っている．したがって，静的なプログラムと動的なプロセスの間のギャップを小さくすることに努力すべきである．」

高校までの数学には，答を得るための手順として，たとえば次のようなものが現れている．これらは，教科書に自然言語で記述されているが，もちろんアルゴリズムである．

2桁以上の数のかけ算，わり算
2次方程式の解法
連立1次方程式の解法
最大公約数の計算

しかし，コンピュータの実行を前提としたアルゴリズムは，動的な実行を意味するものであり，人間に説明し，人間に計算させるためのアルゴリズムとはやや異なるものであ

る．このことを説明するために，頭に思い浮かんだアルゴリズムがコンピュータ向けのプログラミング言語に変換されるまでのアルゴリズム表現の変遷を考えてみる．

（ア）頭にある，もやもやしたアルゴリズム：まだはっきりと整理がされていない思い浮かんだままの手順．

（イ）数式，文章で表されたアルゴリズム：問題解決の手順を，説明のため，整理のために記述したもの．たとえばNewton-Raphson法による方程式の解法は，以下の漸化式で表されるが，これがこの段階でのアルゴリズム表現の例である．

$$X_{n+1}=X_n-\frac{f(X_n)}{f'(X_n)}$$

（ウ）チャート，疑似言語で表されたアルゴリズム：プログラム設計段階の表現である．Newton-Raphson法による方程式の解法は，コンピュータの実行を前提としたこの段階のアルゴリズム表現では，変数としてXだけを使って，以下のように表される．

$$X\leftarrow X-\frac{f(X)}{f'(X)}$$

（エ）プログラミング言語で表されたアルゴリズム：アルゴリズム表現の最終段階であり，コンピュータが実行可能な表現である．

前半の（ア）（イ）段階に比べて，（ウ）（エ）段階のアルゴリズムは，データをどう記憶させながら，どういう順序で実行させるかを明確にしていかなければならない．日常行動や作業手順を表すものをフローチャートに描くことができても，コンピュータに実行させるためのアルゴリズムは必ずしも同じ様な感覚では描けない．１ステップ毎にメモリの内容（すなわち状態）がどう変化すべきかを想像しながらアルゴリズムを作っていかなければならないからである．

(2)（数学の変数とも異なる）プログラムの変数をどう利用するか．：　数学における変数は，普通，ある性質を持った数（などの対象）を代表したものであり，それがどこに記述されていても同じものであることが前提とされている．それに対しプログラムでは，変数は数（などの対象）を格納する場所であり，その中身を変えながら処理を実行することを積極的に行なっていかなければならない．(2)

たとえば，数学における連立方程式

$$2X+\ \ Y=3$$
$$X+3Y=5$$

においては，１行目のXと２行目のXは同じ値を表しているのが大前提であるのに対し，FORTRANなどのプログラミング言語で記述された

$$Y=3-2*X$$
$$X=5-3*Y$$

では，１行目のXと２行目のXではその内容が異なったものになる．

変数の使い方によってアルゴリズムが異なったものになる例をあげよう．

［例１］３つの値　体重A，体重B，体重Cの最大値をMAXに格納するアルゴリズムを作れ．

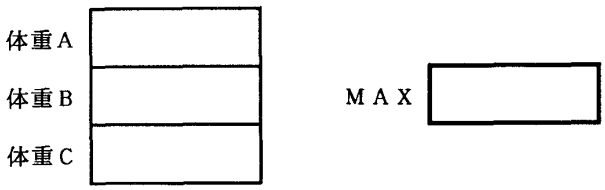

図１　例１の変数

この問題で必要な変数は図－１に示されるものだけである．図－２のアルゴリズムは，まずA，Bの比較をし，Aが大きいときはAとCを比較し，大きい方（AかC）をMAXに代入するものである．このアルゴリズムでは，A，B，Cの大きさの関係によって分類し，変数MAXには代入が１度だけ行われるようになっている．

図－３のアルゴリズムは，MAXに最初A

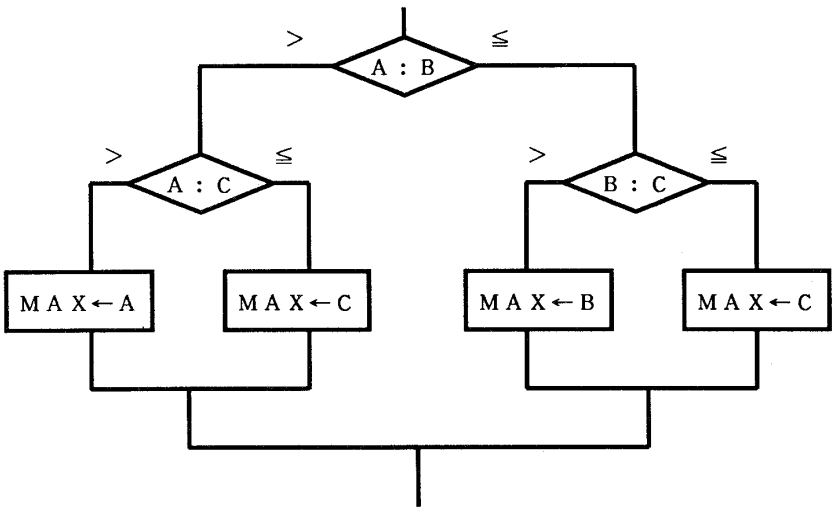

図2　静的アルゴリズム

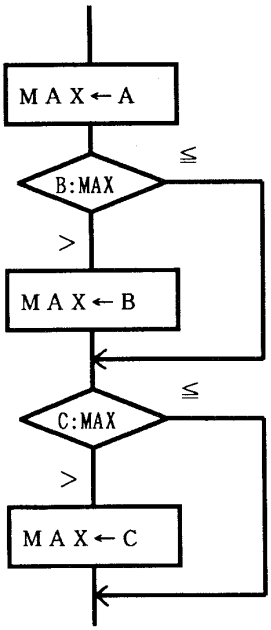

図3　動的アルゴリズム

の値を代入しておき，ＭＡＸとＢを比較し，大きい方をＭＡＸに代入し次にＭＡＸとＣを比較していくものである．このアルゴリズムでは，変数ＭＡＸの値が次々に変化していくようになっている．

プログラミングを習い始めて半年ほどの学生では，ほぼ８割以上が図－２のアルゴリズムを作ろうとする．図－３のような代入によって変数の内容を変化させるような使い方が考えられれば，アルゴリズム作りも一歩力がついたということができる．

(3)　データ操作と制御という２つの異なった機能で分析すること．：　基本的にコンピュータがその記憶状態，外的な状態を変化させるのは入力，計算（代入），出力であり，そのほかのgoto文，for文，if文等は実行の流れを制御するものであり，初心者にはこの２つを整理してアルゴリズムとして表現するのが容易ではない．特にプログラミング言語においては，命令文としてほぼ同じ様な表現で記述しなければならないのが混乱のもとであると思われる．

［例２］多数の製品について，番号と重量を入力し，重量＜100なら不良品の表示をして，最後に不良品の個数を出力せよ．入力のストッパは番号＝999とする．

この問題で必要となるデータは図－４に示される３つである．また，それらの変数に対しデータ操作をどのように行うかを考えると，図－４右の各命令文が得られる．

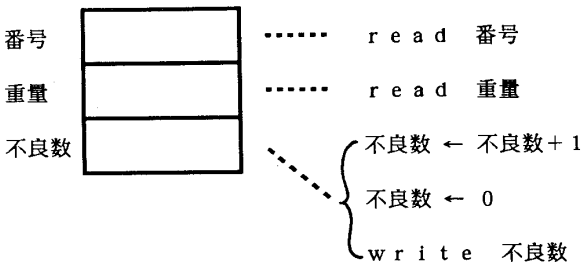

図4　例２のデータ操作

最初から必要な変数，データ操作命令文がこのようにヒントとして与えられていれば，どのような条件の時に実行するか，どのように繰り返すか，等を考えて制御構造を作り易

い．しかし，初心者にとってデータ操作と制御を分離して，順序だててアルゴリズムを構成していくことは容易ではないようだ．

ところで，このようにデータ操作と制御を別々に構成しながらアルゴリズムを作り上げる方法論がジャクソン法（ＪＳＰ）である．ＪＳＰでは，プログラム制御構造を作り（図－5），データ操作命令文を列挙し（図－4右），命令文を制御構造へ割付ける，という流れでプログラムを設計していく．すなわち，制御構造が先に作られ，後からデータ操作を加えていく方法である．

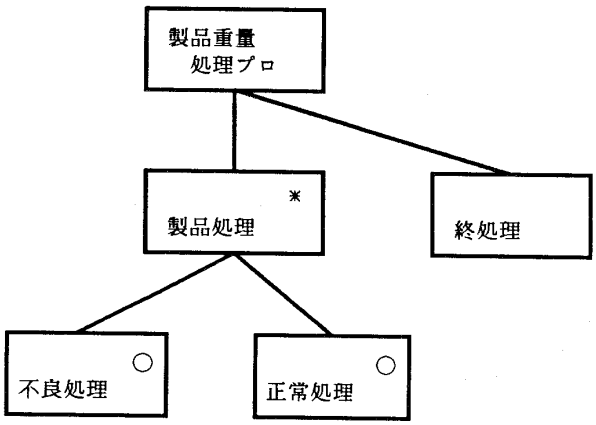

図5　ジャクソン木

(4)　アルゴリズムを作るべき課題そのものがわかりにくい（数学の苦手な学生）．：　数学が苦手な学生には，問題が数式で表現されているだけで抵抗感があり，しかも問題のなかの数式をそのままプログラムに記述出来そうだが，(1)，(2)で見てきたように，必ずしもそう簡単にはいかない．

(5)　学校で学ぶ数学のような「発想のねた（公式）」が明確ではない．：　高校までの数学の問題に対しては，公式に当てはめて式を変形していきながら答を求めることもできたかもしれない．数学の目的はこのようにして問題を解くことだけにあるわけではないが，学生はこうして解けるというだけで一応の目的達成感を抱く．それに対し，アルゴリズム作りを要求する問題は幅広く，既成のアルゴリズムの一部分の書き換えでできてしまうような狭い範囲のものではない．

アルゴリズムを作ることは，数学の証明問題を解くのと同じ様に，全体を組立ていくプロセスを必要とする．しかし，証明問題に使われる公理，定理に相当するような明確化されたものは，現在のところアルゴリズムに対しては作り上げられていない．

(6)　そのほか，数学の問題を解く場合と同様なむずかしさ．：問題を，論理的な関係として分析する能力，問題から本質でないものを取捨して抽象化，記号化する能力，などは数学の問題を考える場合と同じように，アルゴリズム作りに要求されるものである．このような発想のしかたが身に付いているか否かが，アルゴリズム作りにどれほどの抵抗感を抱くかを左右する最も大きな要因かもしれない．

本章で挙げたいくつかの点は，けっして高度な能力，知識を要求しているものとは思えない．しかし訓練と時間を必要とするものである．すなわち，初心者にはゆっくり時間をかけて身に付けさせるように心掛けていかなければならないと考えられる．

4．ま と め

前章で述べたような初心者のつまずきに対し，どのように対応すべきかの特別の具体案をもっているわけではない．初心者には，プログラムの動的性を強調しながら教育することで，自分が何を理解していないかがわかるようにしていきたいと考えている．さらに，構造化プログラミングの意図を示しながら，初心者の段階からよいプログラミングのお作法を身につけるように，今後ともよい表現法，

よい発想法などの，よい道具だてを学生に提供したいと考えている．

文 献

(1) E.W. Dijkstra : Go To Statement Considered Harmful, C. ACM, vol.11, no. 3, pp. 147－148 (1968).

(2) D.E. Knuth : Algorithmic Thinking and Mathematical Thinking, American Mathematical Monthly, March, pp. 170－181 (1985) 邦訳：有澤 誠編，「クヌース先生のプログラム論」, pp. 22－43, 共立出版(1991).