プログラミング言語処理系とアルゴリズム教育

能登 宏

要 旨

この小論は，大学特に文科系大学の基礎的情報処理教育に於いてアルゴリズム教育をどのように進めるべきかを，プログラミング教育の立場から論じたものである．構造化プログラミングとデータ構造を基軸に，アルゴリズム教育ではどのようなプログラミング言語を用いるべきかについて述べる．次に，プログラミング言語処理系によるアルゴリズム教育の具体的な展開についての試論を提示する．事例として，北星学園大学経済学部経営情報学科の基礎的情報処理教育科目に於いて，どのようにアルゴリズム教育が行われているかを示す，併せて最近当学科の授業に於いて実施された，「アルゴリズムの理解度に関する調査」及び，「プログラミングに関するアンケート調査」の分析結果について報告する．

1．はじめに

この研究会の目的は，「大学に於いてアルゴリズム教育をどのように行うべきか？」にあると思われるが，この報告では，

①構造化プログラミングとは何か？

②アルゴリズム教育ではどのような言語処理系を用いるべきか？

③北星学園大学経済学部経営情報学科に於けるアルゴリズム教育についての経験と「プログラミングとアルゴリズムの理解度」についての調査結果の分析

以上の3点について所見を述べ，話題を提供する．

まず，北星学園大学経済学部経営情報学科で用意されている情報処理教育に関するカリキュラムについて述べる．カリキュラムは1，2年次の基礎的情報処理教育と，3，4年次の応用的情報処理教育の2つに大きく分けられる．本学科では，今年度(1992年度)から新カリキュラムが施行されているので，図1には，基礎的情報処理教育の新旧両カリキュラム（教科名，履修年次，単位数，教科内容の概要）を示す．

旧カリキュラムでは，「プログラミング論I」と「プログラミング論II」の言語教育として使用していた言語はそれぞれベーシックとフォートランであったが，新カリキュラムでは「情報処理序論」と「プログラミング論」のいずれに於いてもパスカルとなった．ベーシック，フォートランからパスカルに変更された理由は，アルゴリズムとデータ構造及び構造化プログラミング手法を教育するのに当たって，パスカルが適当であると判断したことによる．この点については，以下の章で詳説する．

以下，章の構成は次のようになっている．

Hirosi Noto 北星学園大学経済学部経営情報学科

＜旧カリキュラム＞（１９９１年度以前に入学した学生に対して）

１年次：「プログラミング論Ｉ（前期４単位）」
初めて情報処理に接する学生を想定して、パソコンの初歩、簡単なプログラミングの初歩（ベーシック）を教育。

「プログラミング論II（後期４単位）」
フォートランによるプログラミング教育

「情報処理概論（通年４単位）」
計算機の構成、ソフトウェアの基礎、情報とデータ、データ通信　など。

２年次：
「プログラミング論III（通年４単位）」
コボルによっ事務処理やシステム設計の基礎を習得する。

＜新カリキュラム＞（１９９２年度から）

１年次：「情報処理序論（前期４単位）」
初めて情報処理に接する学生を想定して、打鍵練習、ワープロ、表計算、簡単なプログラミング論の初歩（パスカル）。

「プログラミング論（後期４単位）」
データ構造とアルゴリズムを一通り習得することを目標とする（パスカル）。

２年次：「計算機科学（通年４単位）」

「応用プログラミング論（通年４単位）」
定型処理のアルゴリズムを構造化プログラミング手法に基ずいて習得するとともにデータベースを利用したデータ処理についても学ぶ。

図１　北星学園大学経済学部経営情報学科の情報処理教育カリキュラムの概要
１，２年次の基礎的情報処理教育（導入的情報処理教育）

第１章「はじめに」
第２章「構造化プログラミングとアルゴリズム」
第３章「データ構造とアルゴリズム」
第４章「プログラミング言語処理系によるアルゴリズム教育」
第５章「アルゴリズム学習の具体的な実習の展開（北星学園大学での事例）
第６章「結論」

２．構造化プログラミングとアルゴリズム

(1)　プログラミング設計に於ける構造化

プログラミング設計に於ける構造化は,「論理の構造化」,「アルゴリズム，或いは，一纏まりの処理手順の構造化」,「コーディングに於ける構造化」とに分けて考えることが出来る．通常,「構造化プログラミング」と言うときには,「アルゴリズムの構造化」を言う場合が多いようであるが，そこに至る「論理の構造化」と，プログラミングの局面での「コーディングに於ける構造化」の２つの構造化が，それと対応して存在している筈である.「論理の構造化」は，あるひとつのまとまった処理手順を考える場合の論理的な制御構造の単位化及び，それらから構成される論理的な複合制御構造の構成のことを言う.「アルゴリズムの構造化」は，このような「論理の構造化」による単位制御構造或いは，複合制御構造を

組み合わせながら，上昇型（ボトムアップ）アルゴリズム或いは，下降型(トップダウン)アルゴリズム又は，両者を併用することによって，階層化された一纏まりの処理手順を構築することを言う．更に,「コーディングに於ける構造化」は,「アルゴリズムの構造化」により階層化された複合制御構造を有する一連の処理手順を，特定の言語処理系を用いてコーディングすることを言う．コーディングに於いてもアルゴリズムの構造化に対応して，上昇型と下降型の構造化されたコーディングが可能である．この場合，当該言語処理系が備えている制御構造の構造化能力によってプログラムの構造化率が大きく左右されることに注意しなければならない．

一般的に「構造化プログラミング」は次のような特徴があると言われている．

・設計し易い
・デバッグし易い
・プログラムを書いた人以外の人にとっても理解し易い．
・アルゴリズムの理解が容易
・プログラムの修正・拡張が容易

「構造化プログラミング」がこのような特徴を持つ理由は，

・段階的詳細化（大きな問題を小さな問題に分割し，小さな部分から大きな部分へ，要素的な部分から全体構成へ）を実現できる
・モジュールの独立性
・GO TO 文のない制御構造

等を挙げることが出来る．

(2) 構造化アルゴリズム

一つの纏まった処理の手順を書き表す場合，大きく分けると次の2つの方式がある．

i）上昇型（ボトムアップ）アルゴリズム
ii）下降型（トップダウン）アルゴリズム

「上昇型（ボトムアップ）アルゴリズム」は，先ず，細部の部品（モジュール）を用意又はあるものと想定しておいて，それらを組合わせ統合化して，大きな全体のプログラムを作ろうとするものである．これに対して,「下降型（トップダウン）アルゴリズム」は，先ずプログラム全体の大枠を定め，次に各部分を順次詳細化して行き，最後に最も下位の部品を作成しようとするものである．いずれのアルゴリズムに対しても構造化の手法が採られる．

実際的には，問題に応じて適宜，2つのアルゴリズムを使い分けたり，両者のアルゴリズムを併用した手法も行われる．問題によっては，強いてこのような構造化手法をとらなくても良い場合もある．特に数値計算上のアルゴリズムにそのような例が多いように思われる．一方，定型的な事務処理のアルゴリズムには，このような構造化されたアルゴリズムが有効である．例を挙げれば，

定型処理：整列処理，制御合計，表引き，
ファイル処理(突合わせ処理)

等である．

(3) 構造化プログラミングのためのアルゴリズム表現法

アルゴリズムを表現するにはいくつかの方法がある．以下に3通りの方法を挙げてあるが，問題によって最適な（或いは，より好ましい）アルゴリズム表現法を選ぶのが良い．

i）流れ図
・定型処理に適している．
「流れ図は出来上がったプログラムを説明するのには役に立つが，これから新しいプログラムを作成するときには，流れ図の作成は余り推薦できない.」という評価もあるが，導入教育では流れ図による視覚的な説明は必要であると考えている．流れ図だけで十分という訳ではないが．

ii）擬似コード
・数学的算法，文字列処理等では，流れ

図よりも有効な場合がある.

・プログラミング言語によるコーディングそのものが擬似コーディング風の表現になっている場合もある.(パスカルによるコーディングはそのような表現に近いと思われる)

iii) 自然言語

・アルゴリズムを自然言語による論理構造として(概念的に)記述する.

・導入的情報処理教育では,学生にアルゴリズムに対して苦手意識や距離感を与えないように,自然言語によるアルゴリズムの記述を有効に利用できる局面もある.

・自然言語による記述法は,数学的算法,定型処理等に効果的に使用できる.

(4) 流れ図[1)]

アルゴリズムの記述方法として図形を用いて視覚的に表現できる「流れ図」を特に取上げてみよう.

i) 流れ図を作成する際の構造化処方

階層構造化されたアルゴリズムを図形を用いて視覚的に表示するにはいくつかの表現形式がある.

・流れ図

・構造化チャート(NS[ナッシー,シュナイダーマン]チャートなど)

・木構造チャート(PADなど)

どの手法を用いるかは,それぞれの手法の特徴を把握した上で,取扱おうとしている問題,或いは,構築しようとしているシステムに適したものを標準化すれば良い.「流れ図」は,処理手順がそのまま図示されるので理解が容易であるが,制御構造が見にくいと言われる.記述の自由度が高く修正は容易である.「構造化チャート」は,任意に制御を移行できない表現形式になっているためアルゴリズムは必然的に構造化される.しかし,書く手間がかかり,修正が難しい.「木構造チャート」は,構造化プログラミングの制御構造を明確に表現できるが,流れ図と違って処理の流れを直接表すものではない.

筆者は,導入的或いは,基礎的情報処理教育に於いては,「流れ図」によるアルゴリズム表現形式が好ましいと考えている.基礎的情報処理教育では,プログラムによるデータ処理の流れに対する直感的理解が,論理的,或いは,概念的な制御構造の把握と平行してどうしても必要であると考える.上記のチャート,特に木構造チャートを描くことは,擬似コード,或いは,プログラミング言語(特に構造化プログラミングに適した言語)による記述と殆ど同じである.即ち,コーディングと一意に対応している.言換えれば,「図形或いは,チャートという形式を採ったコーディングそのもの」と考えられる.そして,上記のチャートを描くには,一定の(変換)規則を,実際の処理の流れと対応させて学ばなければならない.構造化チャートや木構造チャートを説明する場合には,通常の「流れ図」を使用して説明する必要があるのではないだろうか?それならば,まず「構造化された制御構造」と「通常の流れ図」との対応関係を図形認識として明確に把握させ,次に,構造化プログラミングのコーディングに入った方が効果的なように見える.

但し,先にも付言したように,どのような手順或いはアルゴリズムを考えるときにも流れ図が有効かどうかは取り扱っている問題による.

ii) 4つの制御構造(論理構造)の説明

構造化アルゴリズムは,基本的な4つの論理的な制御構造の階層的な組合わせによって構築される.従ってアルゴリズムの構造化のためには,先ず以下の4種類の制御構造をアルゴリズム構築の論理的な構造単位(基本制御構造)として列挙すると同時に,それに付随する流れ図の型を確実に識別しておく必要

がある．

・単純構造
・分岐構造
・反復構造（繰り返し構造）
・多重分岐構造

iii）「流れ図」と「コーディング」との対応の学習

「流れ図」はアルゴリズムを処理手順として描いたものであるが，流れ図を直ちにプログラミング言語によってコーディングすることは，初めてプログラムを書く学生にとっては戸惑いがあるように見える．それは，コーディングそのものが難しいと言うよりも，「流れ図」と「コーディング」との対応が正確に付いていないことに起因していると思われる．「流れ図」で表現された制御構造のパターン，特に基本制御構造についての図形としての型を，先ず明確に識別させる必要がある．その上でそれらの基本制御構造を正確に「コーディング」させる．次にその逆，即ち，「コーディング」から「流れ図」を描くようにする．

この双方向の練習を，基本制御構造と複合制御構造について行うようにする．

この場合次のことに留意する．「流れ図」は記述の自由度が高いために，同一の制御構造でも一見異なる図形パターンに見えることがある．「流れ図」の描きかたの標準を定めておくと好い．

(5) 構造化プログラミングに適した言語

コーディングする際に使用するプログラミング言語について考える．構造化されたアルゴリズムが出来上がれば，後はどのようなプログラミング言語を用いても同等にコーディングが出来るのであろうか．

i）制御構造を記述するのに適した言語がある

制御構造，特に構造化された制御構造を記述する場合，言語によって，コーディングし易いものとしにくいものがある．コーディングし易い言語とは次のような条件を満たしている言語であろう．

1．関数や手続きなどの副プログラムを主プログラムや他の副プログラムとは独立に定義することが出来る．
2．大域的変数と局所的変数の区別がある．
3．基本制御構造に直接的に対応する命令系が用意されている．
4．行番号という概念が基本的にない．

現在標準的に使用されているプログラム言語についてこれらの条件が満足されている度合いを比較してみる．

パスカル，C ❶手続きを主プログラムや他の副プログラムと独立に定義出来る．
❷大域的変数と局所的変数との区別がある．
❸制御構造に対応した命令群がある．
❹行番号という概念が基本的にない．
【評価】構造化プログラミングに適している．

コボル ①手続きは定義できるが，主プログラムと独立ではない．
2 大域的変数と局所的変数との区別がない．
❸制御構造に対応した命令群が整備されている．
❹行番号という概念はあるがコーディングでは使用しない．
【評価】構造化プログラミングは可能である．

フォートラン ❶手続きを主プログラムや他

の副プログラムと独立に定義出来る.

❷大域的変数と局所的変数との区別がある.

③制御構造に対応した命令群が貧弱ある.

4 行番号を使用しなければならない.

【評価】構造化プログラミングは苦手である.

ベーシック　①手続きは定義できるが，主プログラムと独立ではない.

2 大域的変数と局所的変数との区別がない.

③制御構造に対応した命令群が貧弱である.

4 行番号を使用しなければならない.

【評価】構造化プログラミングに馴染まない.

ここで，❶，❷……の項目は，完全に或いは，ほぼ条件が満たされていることを示し，①，③，……の項目は，余り条件が満たされていないことを示し，*2*，*4*，……の項目は，条件が完全に或いは，殆ど満たされていないことを示している．上記の評価によれば，構造化プログラミングに最適な言語は，パスカル又は，C言語，おおよそ適しているのがコボル言語ということになる．従来からよく用いられてきているフォートランやベーシック言語は，構造化プログラミングには余り適していないと言える.

以下に基本制御構造に対応した命令群の一覧をコボルとパスカルについて示す.

【例】コボル言語

・分岐構造

IF....THEN....ELSE....END-IF.

・反復構造の記述

PERFORM 文（各種書き方の自由度がある）

・多重選択構造

EVALUATE....WHEN....END-EVALUATE 文

パスカル言語

・分岐構造

IF....THEN....ELSE.... ;

・反復構造の記述

FOR....DO，WHILE....DO，

REPEAT....UNTIL.... ;

・多重選択構造

CASE 文

ii）定型処理に対応したアルゴリズムが組込まれている命令群が用意されている言語

コボル言語はもともと事務処理用言語として設計された言語処理系であるから，事務処理上よく用いられる定型的な処理については，コボル言語の命令群として予め組み込まれている.

【例】コボル言語に於ける定型処理の命令

整列処理

SORT　ファイル名

昇順（または降順）　キー項目

USING　入力用ファイル

GIVING　入力用ファイル.

探索処理（表引き処理）

SEARCH　表の名

VARYING　検索の指標

AT END.....

WHEN　検索条件

END-SEARCH.

これは，アルゴリズムの構造化の過程に於いて，「整列」とか「検索」という一纏まりの処理手順が，言語処理系の「命令」によって，単なる一つの構造単位となってしまったことを意味している．定型処理の「命令化」によって構造化プログラミングは更に見易く且つ理解し易いものになるのは当然のことである.

iii）数値計算のアルゴリズムが組込まれている関数群が用意されている

ii）で述べたことと同様のことが数値計算向きの言語に於いても行なわれている．各種言語処理系に予め組込まれている関数や手続きの一群がある．それらはあたかも一つの命令としてコーディングの中で使用することが出来る．

【例】パスカル，フォートラン，ベーシック，C言語
乱数発生，最大値，最小値の判断等々

これらは，主として数学に於ける関数的なものであり，引き数によって引用し処理できるデータ量も極めて少ないのが通例であるが，アルゴリズムの構造化の単位として寄与する．

iv）構造化プログラミングがアルゴリズムの理解を容易にしている事例

構造化プログラミングの手法はアルゴリズムの理解を助ける．それは構造化されたアルゴリズム（実際には流れ図で描かれている場合が多い）を，そのまま1対1に対応した構造単位を用いてコーディング出来るからである．その場合構造化アルゴリズムの流儀に従って，

a）トップダウンとボトムアップ方式のプログラミング
b）トップダウンとボトムアップの方式を適宜使い分けるプログラミング

の2通りの手法が考えられる．

基礎的情報処理教育課程で，構造化プログラミングを教える場合には，a）の方法ではなく，b）の方法によった方が好いようである．初心者にあっては，コーディングするのに先立って，プログラム全体の構成の見通しがきくと同時に，コーディングの際必要とされる個々の部品が始めから見えていた方が，プログラミングに対する描像が具体化し易いように見えるからである．構造化プログラミングの簡単な例は，後の章で具体的に示される．

v）構造化プログラミング言語によって，アルゴリズム習得上の本質的な問題点が解消する訳ではない

今まで述べて来たことと抵触しかねないやや逆説的な見出しである．後の章で紹介するように，学生のアルゴリズムの理解度という点に関しては極めて厳しい問題が現実には存在する．

構造化プログラミングは，構造化されたアルゴリズムを的確に表現するし，又，それを用いることによって学生のアルゴリズムの把握が容易になることは確かなことである．しかし，プログラミングすべきある具体的な問題が与えられたときに，コーディングに先立って，学生が独力でアルゴリズムを考え，或いは，構造化アルゴリズムを発想出来るかと言うと，後の調査結果からも分かるように，実際にはそうなっていない．

つまり，構造化されたアルゴリズムの表現の容易さ・表現の見易さと，アルゴリズムの構築・アルゴリズムの創造との間には小さからぬ間隙或いは，障壁が存在するように思われる．この間隙や障壁となっているものは何かと言う問題である．問題と思われる事項を列挙してみる．

a）問題として要求されている処理内容は何なのかを明確にする．これはコーディング以前の問題であるがおろそかに出来ない．
b）連続的に進んでいるように見える処理手順から，どのようにして「処理の各段階」を切出し，一つの段階から次の段階に「移行する契機（条件）」を見出すか．
c）切出された「処理の各段階」及び「移行の契機（条件）」をアルゴリズムとしてどのように表現すれば良いのか．
d）手順をどのように構造化すれば良いの

か.

e）アルゴリズムを表現するためのプログラミング言語に関する知識が十分かどうか.

f）構成されたアルゴリズムを，与えられたデータ構造(変数)を用いてコーディング出来るか.

アルゴリズム教育の中には，このような問題が内在している．例えば「整列処理」とか「検索処理」のアルゴリズムをコボル言語を使用して表現することを考えて見る．コボル言語では既にそのような処理機能が命令の一つとして組込まれているから,「整列処理」とか「検索処理」のアルゴリズムの問題は克服されたのかというとそうではない．特定の処理手順が命令化されることによってなされたことは，アルゴリズム（或いは，コーディング）の段階的詳細化が最下部まで行かなくても良くなったということであって，これらのアルゴリズム習得上の問題は，それぞれの詳細化及び構造化のそれぞれの段階に依然として存在していると考えなければならない.

更に，アルゴリズム固有の問題として，発展的な課題にも留意すべきである.

a）例えば，コボル言語に於ける「SORT 命令」が，どのようなアルゴリズムを利用した整列処理を行っているかを知ることは有用である．問題に即した最適の整列処理（最もデータ構造に見合ったアルゴリズム，最も計算量の小さいアルゴリズム）を選択する事が出来るようになるからである.

b）一般的に言って，問題解決，或いは，データ処理の仕方には最適のアルゴリズムが存在すること，そしてそのアルゴリズムを採用することの利得を理解させる必要がある.

3．データ構造とアルゴリズム

処理すべきデータがどのような構造（型と配置）に従って格納されているかは，アルゴリズムを考えるときに欠かせない処理条件である．つまり，アルゴリズムはデータ構造抜きには考えられないし，又データ構造はアルゴリズムに適したように設計されなければならない.

(1) データ構造とプログラミング言語

プログラミング言語処理系には，構造を持ったデータ型を自由に定義できるものがある．このような言語は，処理内容に見合った最適のデータ構造を用意する上で好ましいプログラミング環境を提供する．構造を持ったデータ型は，レコード型とか構造体とか呼ばれている．構造を持ったデータ型を許している言語には，パスカル，C，コボルなどがある．フォートランやベーシックにはこのような型は定義されていない.

i）構造を持ったデータ型

・レコード型　　PASCAL

・構造体　　C

・レコード記述項　　COBOL

ii）新しい型を定義できる言語処理系

PASCAL，C，COBOL

(2) データ構造に対応したアルゴリズム

データ構造は問題を解決しようとするときの思考過程（最小の処理時間，簡単なデータ入力，無駄な繰返しのないデータ処理なども含めた）に密接に関係している．処理内容とデータ構造の対応の例は例えば次のようなものである.

【例】・整列，探索処理のアルゴリズムに対応したデータ構造

スタックとキュー

木構造

リスト構造

・ファイル編成に基ずいたファイル操作のアルゴリズムに対応したデータ構造

レコード型，レコード記述項

このように，構造を持ったデータ型を許容しているプログラミング言語処理系を使用すると，著しくアルゴリズムの構築と理解が容易になる．

4．プログラミング言語処理系によるアルゴリズム教育

では，プログラミング言語処理系によるアルゴリズム教育の実際はどのようにあるべきであろうか．

(1) プログラミング言語処理系によるアルゴリズム教育の実際

上で述べたように，論理の構造化を重視し，アルゴリズムの構造化を行う．構造化されたアルゴリズムの視覚的表現形式として適当な「流れ図」を採用する．コーディングに当っては，極力構造化プログラミングが可能な言語処理系を選択することがまず提起されなければならない．

しかし，構造化プログラミング言語処理系を採用しても，それは構造化アルゴリズムの理解を容易にする為の必要条件である．具体的なコーディングに入る前提として，以下の内容を意識的に演習をしておいた方が良いと思われる．

i）簡単な例題を取上げて，論理的な基本制御構造を，対応する「流れ図」と共に理解させる．

ii）次に，簡単な処理手順を，基本制御構造が組込まれた複合的制御構造を用いて表現する．例題としては，身近な事象でも好いし，数学的問題でも好いし，ある定型的な処理でも好い．

iii）上のような演習によって，処理手順と制御構造の型（タイプ）を「流れ図」を通して視覚的に把握させることが重要である．処理手順を「流れ図」で表すだけでなく，問題によっては，擬似コードで表現したり，或いは，普通の言葉で表現したりするのも教育的である．

iv）このようなアルゴリズム構築の基礎的な演習を前提として初めて，構造化されたコーディングの作業が意味を持ち且つ有効になる．コーディングの演習に当っては，「流れ図」からコーディングすることと，逆にコーディングから「流れ図」を描くことの双方向の練習が有効なようである．

アルゴリズムの演習では，「制御切れ・制御合計」とか「突き合わせ」などの定型的な処理とか，「探索」・「整列」などのような基本的且つ代表的な幾つかのデータ操作について，その標準的なアルゴリズムの型（タイプ）を覚えることが特に有効である．

いずれにしても，アルゴリズムとか「流れ図」の型（タイプ）を視覚的に覚え，その制御構造をいろいろの例題に適用したり応用したりすることが，最終的にはアルゴリズムの創造へと発展すると思われる．

(2) 応用ソフト利用時に於ける"各種言語"によるアルゴリズム構築の必要性

次のようなことを，プログラミング言語を用いてアルゴリズムを学ぶ必要性の一つとして提示するのも，学生にとってはアルゴリズム学習の一つの動機付けとなるであろう．

与えられた問題を解決しようとするとき，当該問題に固有のアルゴリズムが求められることがしばしばある．言語処理系に準備されている"標準的な"「手続き」や「命令構文」の処理内容と若干異なる処理を行なわなければならないとか，「手続き」の処理内容を越えて，処理の手順を考えなければならないよう

な場合である．このことは，如何に定型処理機能を組込んだ命令群が準備されていても，或いは，もっと一般的に言って，如何に市販の汎用的な応用ソフトが準備されていても，「我々がある種の"言語"を用いてアルゴリズムを考なければならない」状況に十分，或いは，それなりに対応出来るようになっていることが好ましい．

(3) 構造化されたアルゴリズムの階層的発展への展望の提示

我々はここ迄のところ，専ら単一プログラムに於ける構造化アルゴリズムを考えて来た．しかし構造化プログラミングの手法により表現される構造化されたアルゴリズムは，単一プログラムから重層的或いは，階層的に発展し，「構造化設計手法」としてシステム設計に直接つながって行く論理性を持っている．従ってアルゴリズムをプログラミング言語を学ぶ為の制御構造として捉えるばかりではなく，プログラミング言語による導入的アルゴリズム教育の発展的な展望も具体的に提示し，学生のアルゴリズムに対する興味と意欲を喚起し，動機付けとしたい．

5．アルゴリズム学習の具体的な実習の展開（北星学園大学での事例）

北星学園大学経済学部経営情報学科では，本年度（1992年度）から「情報処理序論」と「プログラミング論」では，プログラミング言語としては，パスカルを採用した．理由は，前章で詳しく論じたように，パスカルが構造化プログラミング言語であり，且つデータ構造が自由に定義出来るからである．「プログラミング論III」では，コボル言語を採用している．システム設計の基礎として，定型的な事務処理を行うための機能が用意されていること，(手続きが主プログラムと独立には定義出来ないという点はあるが) 構造化プログラミングが可能なこと，そしてレコード記述の自由度が高いことがその理由である．この章では，アルゴリズム教育として，主として「情報処理序論」と「プログラミング論III」の事例を見てみよう．

(1) プログラミング環境

初めてプログラミング環境に接する「情報処理序論」を履修する学生にとって，ターボパスカルが備えているプログラミングの統合環境は大変有用である．「統合環境」というのは，ターボパスカルが装備している利用者支援環境で，プログラムの編集・実行，ファイル処理等の標準的な機能の他に，デバッグ機能，追跡機能，実行中断機能などがある．プログラムに即したデータ処理の流れを，一命令毎に追跡することが出来るだけでなく，プログラム実行中に，変数の内容を覗いて見たり，実行の処理とは別に，必要な式の評価をする事も出来る．

始めのうちは，少なからぬ学生が，プログラムに於ける変数の概念，入出力処理に於けるデータの流れ，さらには代入文の意味等が把握出来ない．このような知識がないと，データ構造を扱うアルゴリズムの理解と構築は困難なものとなって来る．今年度初めて「統合環境」を利用してみたが，授業後の感想や定期試験の結果から判断すると，例年と比べて，「変数」,「変数と定数の違い」,「変数へのデータの読込みと変数からのデータの書出し」,「変数への値の代入」,「代入文」,「変数の中身の動的（時間的）変化」などの理解が良好のように見える．引き続きこのような「統合環境」のもとでの導入的情報処理教育を実施して行きたいと考えている．

(2) 演習の実際

「プログラミング論III」を例にとるとおおよそ次のような仕方で，実習を伴った授業は進められている．

i）まず基本制御構造を示し，これがアルゴリズムを構築する構造単位となることを説明する．次に，基本制御構造の「流れ図」と「コーディング」との対応を理解させる．コーディングの基本となるこの過程を十分に理解させるようにすることが非常に重要である．

ii）「プログラミング論 III」では，7つの型の定型処理のアルゴリズムを取上げている．

a）簡単な入出力
b）報告書作成
c）制御切れ・制御合計
d）整列処理
e）表引き
f）指標付き表操作
g）ファイルの突合わせ処理

この7つの定型処理を例題として，以下のような演習を行っている．もちろん時間的な理由から全ての場合について練習することは出来ないが．

iii）問題の分析とデータ構造の設計

プログラムに必要にして十分な処理条件と処理手順を記述する．共通に読込むデータが準備済みなので，データ構造は確定している場合が多い．

iv）定型処理の中核をなす制御構造について一通り説明した後，次のような対応

・流れ図 → コーディング
・コーディング → 流れ図

即ち，「流れ図」からコーディングを行い，逆にプログラムから「流れ図」を描く双方向の練習を行う．

(3) 演習時の理解度チェック項目

プログラムの作成にまで至るアルゴリズムの教育には，いくつかのチェックポイントがある．例えば，「プログラミング論 III」では，次のような項目内容についての理解度をチェックをしながら授業を進めている．これによって学生が「『プログラミング論 III』が難しい」と言うときに，問題提示からコーディングまでの行程のどこに問題があるのかを明らかにすることが出来る．

a）問題分析（問題の仕様を明らかにする）の難しさなのか？
b）手順（アルゴリズム）のむずかしさなのか？
c）プログラムの構造化が問題なのか？
d）流れ図の表現の難しさなのか？
e）コーディング（文法）のむずかしさなのか？

これにデータ構造の理解度に関する項目を付加するともっと好いであろう．

(4) 学生のアルゴリズム理解度の調査

(3)まで述べてきた「プログラミング論 III」に於けるアルゴリズム実習を，1）簡単なアルゴリズム理解度試験による客観評価と，2）学生に自己申告させるアンケート調査の2つの面から評価して見よう．

i）アルゴリズム理解度調査と調査結果

「アルゴリズム理解度調査」が，本年度(1992年度）の「プログラミング論 III」履修学生4組129名を対象にして，11月30日（B，C，D組）と12月2日（A組）に行われた授業時間を割愛して実施された．この理解度調査は，「基本制御構造」と「階層化された複合制御構造」の理解度を，(A)「流れ図」から「コーディング」及び(B)「コーディング」から「流れ図」への処理手順の表現形式の変換操作を通じて測定しようとするものである．「アルゴリズム（手順）の理解に関する調査」の調査用紙を図2に示す．I．基本制御構造(A)は4問からなり，①順次構造，②選択構造，③反復構造，④多重選択構造についての「流れ図」をコーディングさせている．(B)では逆に，①から④の制御構造を含むプログラムの一部をそれぞれ「流れ図」に直させている．II．データ構造は，複合制御構造について問うている．①では「流れ図」をコーディングさせている．②では，複合制御構造を含むプログラムの一

アルゴリズム（手順）の理解に関する調査 （採点基準） プログラミング論Ⅲ(A, B, C, D) 番号(　　) 名前(　　)

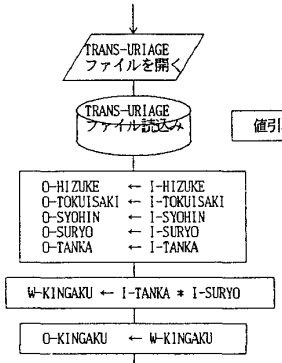
この調査は、アルゴリズムの理解度を、「コーディング」と「流れ図」の両面から調査しようとするものです。
「プログラミング論Ⅲ」の成績評価とは全く関係がありませんので、普段の実力を十分発揮して下さい。

100点 満点

Ⅰ．基本的な制御構造（制御構造の単位） 55点

（A）次の流れ図に従ってコーディングをしなさい。 24 点

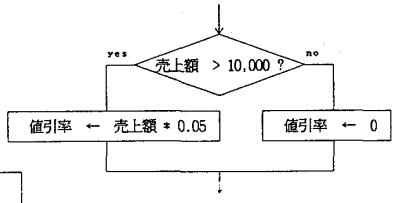
①順次構造



【7 点】

```
(1)  OPEN INPUT TRANS-URIAGE.
(1)  READ TRANS-URIAGE.
     MOVE I-HIZUKE TO O-HIZUKE.
     MOVE I-TOKUISAKI TO O-TOKUISAKI.
(2)  MOVE I-SYOHIN TO O-SYOHIN.
     MOVE I-SURYO TO O-SURYO.
     MOVE I-TANKA TO O-TANKA.
(1)  COMPUTE W-KINGAKU = I-TANKA * I-SURYO.
(1)  MOVE W-KINGAKU TO O-KINGAKU.
(1)  WRITE URIAGE.
```

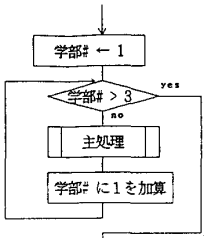
②選択構造



【5 点】

```
(1)  IF URIAGE-GAKU > 10000 THEN
(2)  COMPUTE NEBIKI-RITU = URIAGE-GAKU * 0.05
(1)  ELSE
(1)  MOVE 0 TO NEBIKI-RITU.
```

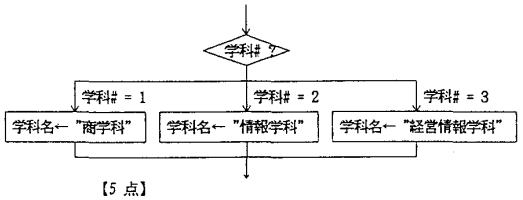
③反復構造



【7 点】

```
(2)  PERFORM SYUSYORI
(1)  VARYING GAKUBU-NUMBER
(2)  FROM 1 BY 1
(2)  UNTIL GAKUBU-NUMBER > 3.
```

④多重選択構造



【5 点】

```
(1)  EVALUATE GAKKA-NUMBER
(1)  WHEN 1 MOVE NC "商学科" TO GAKKA-MEI
(1)  WHEN 2 MOVE NC "情報学科" TO GAKKA-MEI
(1)  WHEN 3 MOVE NC "経営情報学科" TO GAKKA-MEI
(1)  END-EVALUATE.
```

（B）次のプログラムの一部分を流れ図にしなさい。 31 点

①

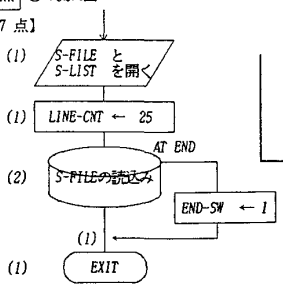
```
    OPEN INPUT S-FILE
         OUTPUT S-LIST.
    MOVE 25 TO LINE-CNT.
    READ S-FILE AT END MOVE 1 TO END-SW.
YOMIKOMI-EXIT.
    EXIT.
```

②

```
    IF ZAIKOSU <= 4000 THEN
      MOVE 6000 TO HACCYU
    ELSE
      CONTINUE
    END-IF.
    WRITE URI-REC.
```

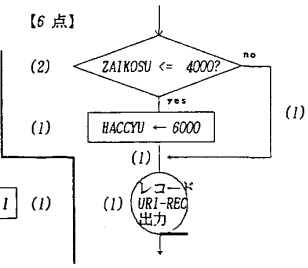
①の流れ図

【7 点】



②の流れ図

【6 点】



③

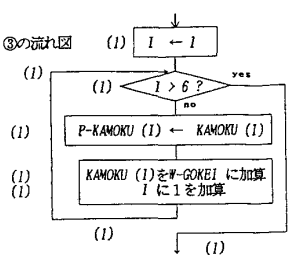
```
    PERFORM VARYING I FROM 1 BY 1 UNTIL I > 6
      MOVE KAMOKU (I) TO P-KAMOKU (I)
      ADD KAMOKU (I) TO W-GOKEI
    END-PERFORM.
    COMPUTE HEIKIN = W-GOKEI / 6.
```

④

```
    EVALUATE DATA-A
      WHEN 1 MOVE 50 TO CHARGE
      WHEN 2 MOVE 100 TO CHARGE
      WHEN 3 MOVE 200 TO CHARGE
      WHEN OTHER MOVE "ERROR" TO FLAG
    END-EVALUATE.
```

③の流れ図

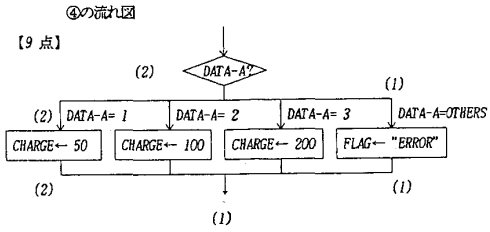
【9 点】



プログラミング論Ⅲ(A, B, C, D) 番号(　　) 名前(　　)

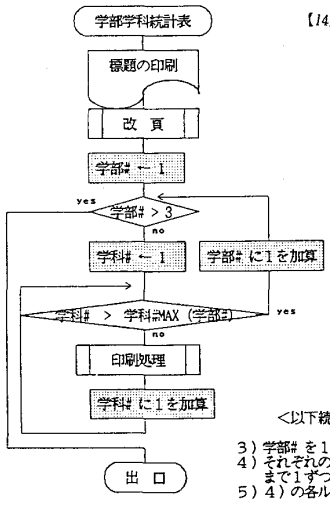
④の流れ図

【9 点】



Ⅱ．組合わされた制御構造（基本制御構造の組合わせ） 45点

① 次の流れ図をコーディングしなさい。 23 点



【14点】

```
     GAKUBU-GAKKA-TOKEI-HYO.
(1)      WRITE HYOUDAI.
(1)      PERFORM KAI-PAGE.
(1)      PERFORM VARYING GAKUBU-NUMBER
(2)          FROM 1 BY 1
(2)          UNTIL GAKUBU-NO > 3

(1)      PERFORM INSATU-SYORI
(1)      VARYING GAKKA-NUMBER
(2)          FROM 1 BY 1
(2)          UNTIL GAKKA-NUMBER > GAKKA-MAX (GAKUBU-NUMBER)

(1)      END-PERFORM.
     GAKUBU-GAKKA-TOKEI-HYO-EXIT.
         EXIT.
```

<以下続けなさい>

3）学部# を1から3まで1ずつ増加させるループを回す。 (3)
4）それぞれの学部# に対して、学科# が1から学科#MAX（学部#）まで1ずつ増加させるループを回す。 (4)
5）4）の各ループ毎に「印刷処理」を行う。(2)

この流れ図の処理内容の手順を、以下の例にならって、言葉で表現しなさい。

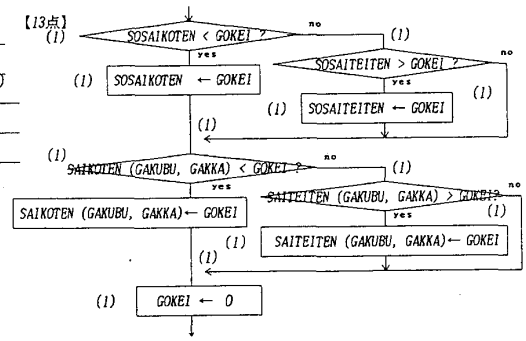
【9 点】
1）標題を印刷する。
2）手続き「改頁」を行う。

4）1）でないとき、「最低点（学部#, 学科#）」＞「合計」ならば、「最低点（学部#, 学科#）」を「合計」で更新する。 (3)
5）「合計」に 0 を転記する。 (1)

②次のプログラムの一部の流れ図を書きなさい。 22 点

```
IF SOSAIKOTEN < GOKEI THEN MOVE GOKEI TO SOSAIKOTEN
  ELSE IF SOSAITEITEN > GOKEI THEN MOVE GOKEI TO SOSAITEITEN.
IF SAIKOTEN (GAKUBU, GAKKA) < GOKEI THEN
                MOVE GOKEI TO SAIKOTEN (GAKUBU, GAKKA)
  ELSE IF SAITEITEN (GAKUBU, GAKKA) > GOKEI THEN
                MOVE GOKEI TO SAITEITEN (GAKUBU, GAKKA).
MOVE 0 TO GOKEI.
```

【13点】



このプログラムの一部の処理内容の手順を、以下の例にならって言葉で表現しなさい。

【9 点】
1）「総最高点」 ＜ 「合計」ならば、「総最高点」を「合計」で更新する。
2） 1）でないとき ・・<以下続けなさい> ・・・・・
1）でないとき、「総最低点」 ＞ 「合計」ならば、「総最低点」を「合計」で更新する。 (3)
3）「最高点（学部#, 学科#）」＜ 「合計」ならば、「最高点（学部#, 学科#）」を「合計」で更新する。 (2)

図2　アルゴリズム（手順）の理解に関する調査

基本制御構造の理解度調査（%）

	順次構造	選択構造	反復構造	多重選択構造	全体
「流れ図」---＞「ｺｰﾃﾞｨﾝｸﾞ」					
A　組（29名）	85.2	75.1	37.9	69.6	66.1
B　組（31名）	69.6	74.8	38.7	69.0	61.6
C　組（31名）	99.1	94.8	32.7	94.2	77.8
D　組（38名）	100.0	95.3	74.4	94.2	90.3
「ｺｰﾃﾞｨﾝｸﾞ」---＞「流れ図」					
A　組（29名）	62.5	74.1	27.2	49.8	50.8
B　組（31名）	76.0	82.8	17.9	71.0	59.0
C　組（31名）	73.5	72.6	44.4	77.4	66.0
D　組（38名）	89.8	88.2	64.9	93.0	83.2

基本制御構造の理解度調査（%）

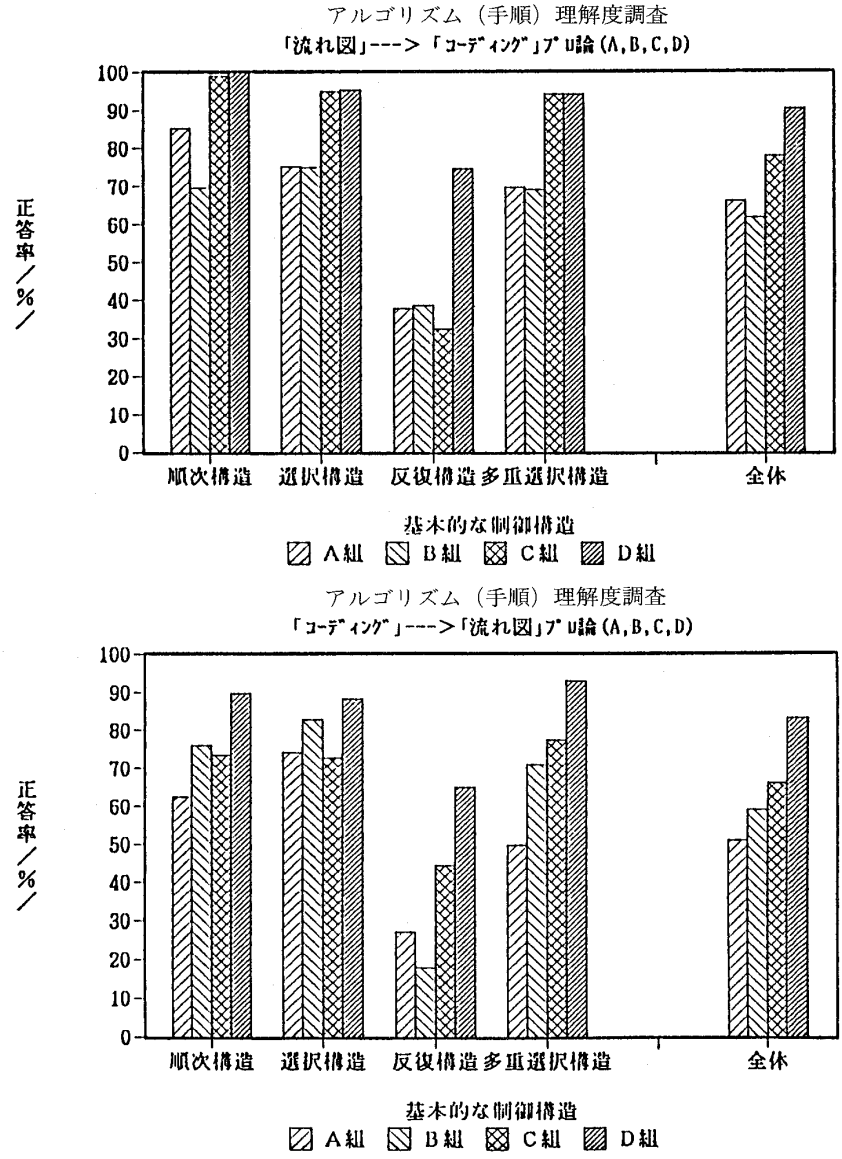


図3　基本制御構造の理解度調査（%）

部に対応する「流れ図」を描かせている．いずれの場合も処理内容の手順を箇条書き風に普通の言葉で書かせる設問も付してある．各問の配点は図2の問題用紙に表示してある通りである．

図3には，Ⅰ．について，①から④それぞれについての正答率と①から④を纏めた正答率をクラス毎に示した．集計結果は表形式のものとグラフ化したものの両方を示した．4クラス併せて，単独の基本制御構造は「反復構造」を除いて，(A)は75%，(B)は66%の正答率を示している．「反復構造」の正答率は，(A)が47.6%，(B)が40.2%である．図4には複合的制御構造に関する結果を示す．4クラス併せての正答率は，①の「流れ図」→「コーディング」については，31%，②の，「コーディング」→「流れ図」については，16%となっている．

	「流れ図」---＞「ｺｰﾃﾞｨﾝｸﾞ」	「ｺｰﾃﾞｨﾝｸﾞ」---＞「流れ図」
A 組 (29名)	13.8	12.3
B 組 (31名)	17.1	15.4
C 組 (31名)	29.7	10.1
D 組 (38名)	56.6	24.6

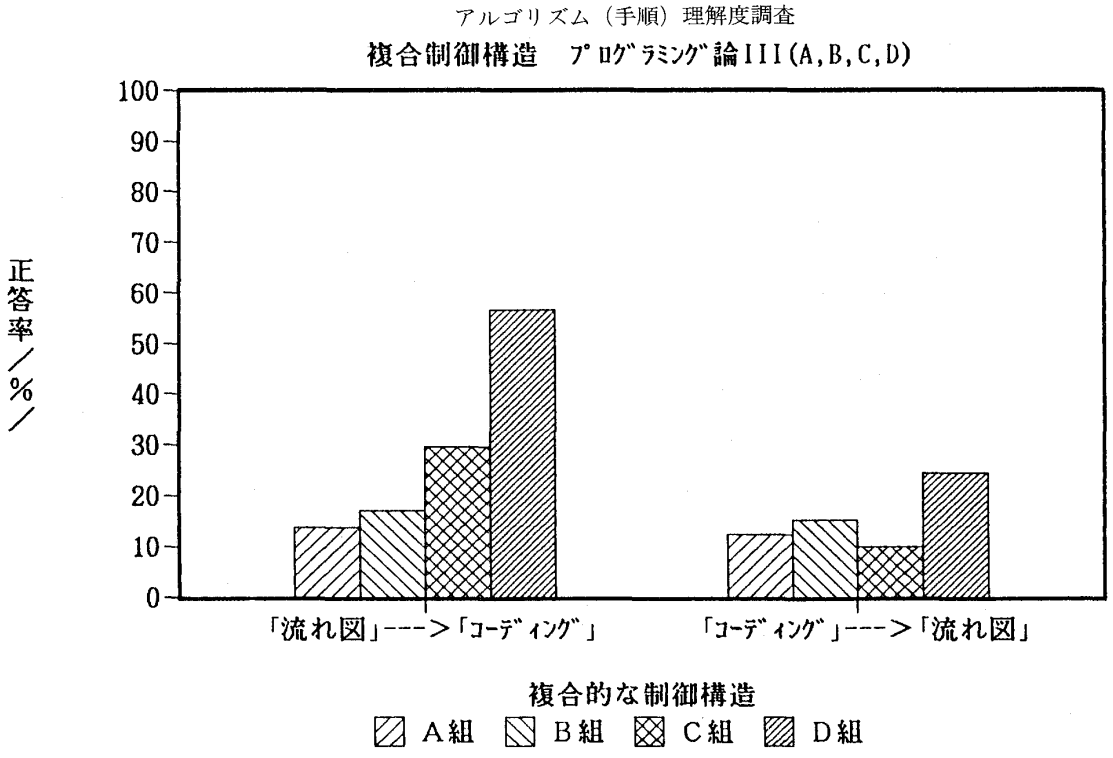


図4 複合的制構造の理解度調査（%）

ii）学生の自己申告によるアンケート調査

次に学生の「自己申告」による「プログラミングに関する質問紙調査」について報告する．この調査は，本年度(1992 年度)の「プログラミング論 III」履修学生 4 組併せて 126 名を対象にして，12 月 2 日（A組）と 12 月 7 日（B，C，D組）に行われた授業時間を割愛して実施された．調査用質問紙を図 5 に示す．同図には便宜上，集計結果も既に挿入してある．設問は，3 つの大きな項目に分かれている．(1)構造化プログラミングについて，(2)アルゴリズムとプログラムについて，(3)アルゴリズムとプログラム言語についてである．これらの大きな項目は，数個の小項目からなり，全体で 13 の設問がある．

特徴的な結果を列挙する．なお，100 分率は有効度数を 100%としている．

(1)構造化プログラミングについて

①構造化プログラミングについては，3 割の学生が「おおよそどのようなものかは理解できる」と答えているが，②「構造化プログラミングが分かりやすい」と答えた学生は 15%であり，大半の 70%の学生は，「どちらとも言えない」と答えている．③「構造化プログラミングのどのような点が分かり易いか」（多肢選択）という設問に対しては，「処理手順を分割して考える事が出来る」(33%)「プログラムが見易い」(24%)「プログラム全体の見通しが良い」(21%)「流れ図との対応が取り易い」(20%) となっている．④「構造化プログラミングが分かり易い理由」（多肢選

プログラミングに関するアンケート　12／2／1992
プログラミング論Ⅲ(A, B, C, D)　番号(　　)　名前(

次の各項目に答えて下さい。　<<　回答数　A組（29名）B組（30名）C組（31名）D組（36名）合計 126名　>>

（1）構造化プログラムとアルゴリズムについて

①構造化プログラミングとはおおよそどのようなものかは理解出来る。
設問　1　（有効度数を100 に換算）
　1．よく理解出来る　2．まあまあ理解出来る　3．どちらとも言えない　4．あまり理解出来ない　5．殆ど理解出来ない

階級　0（ 0）:　|
階級　1（ 0）:　|
階級　2（ 29）:　| ****************************
階級　3（ 33）:　| *********************************
階級　4（ 29）:　| ****************************
階級　5（ 9）:　| *********

②構造化プログラミングは分かり易い。
設問　2　（有効度数を100 に換算）
　1．大いに賛成　　2．賛成　　3．どちらとも言えない　4．反対　　5．大いに反対

階級　0（ 1）:　| ?
階級　1（ 0）:　|
階級　2（ 15）:　| ***************
階級　3（ 70）:　| **
階級　4（ 13）:　| *************
階級　5（ 2）:　| **

③構造化プログラミングのどのような点が分かり易いと思いますか？（多肢選択可）
設問　3　（有効延べ度数を100 に換算）
　1．プログラムの設計がし易い　　5．プログラム全体の見通しが良い
　2．アルゴリズム（手順）が理解し易い　　6．プログラムが見易い
　3．処理手順を分割して考える事が出来る　　7．デバッグ（誤り探し）がし易い
　4．流れ図との対応がとり易い　　8．プログラムの修正が簡単

階級　0　（ 12）:|　????????????
階級　1　（ 9）:　|　*********
階級　2　（ 9）:　|　*********
階級　3　（ 24）:|　************************
階級　4　（ 14）:|　**************
階級　5　（ 15）:|　***************
階級　6　（ 17）:|　*****************
階級　7　（ 6）:　|　******
階級　8　（ 6）:　|　******

図5　プログラミングに関するアンケート調査

④構造化プログラミングが分かり易い理由は何だと思いますか？（多肢選択可）
設問 4 （有効延べ度数を100 に換算）
1．問題を大きな部分から小さな部分に分割してプログラム出来る
2．部品を用意してそれを組み合わせてプログラムを構成出来る
3．段落（或は、手続き）をそれぞれ独立に書くことが出来る
4．制御構造が書き易い
5．プログラム全体の制御構造の見通しが良い

```
階級 0 ( 17 ):| ??????????????????
階級 1 ( 27 ):| ****************************
階級 2 ( 19 ):| ********************
階級 3 ( 39 ):| ****************************************
階級 4 ( 2 ): | **
階級 5 ( 14 ):| **************
```

（2）アルゴリズムとプログラミングについて

⑤プログラムを作る際にヒントがないとプログラム出来ない。
設問 5 （有効度数を100 に換算）
1．大いに賛成 2．賛成 3．どちらとも言えない 4．反対 5．大いに反対

```
階級 0 ( 0 ):  |
階級 1 ( 48 ): | ************************************************
階級 2 ( 39 ): | ****************************************
階級 3 ( 13 ): | *************
階級 4 ( 0 ):  |
階級 5 ( 0 ):  |
```

⑥プログラミングをする場合に、流れ図は有効だと思いますか？
設問 6 （有効度数を100 に換算）
1．大いに有効 2．有効と思う 3．どちらとも言えない 4．有効とは思わない 5．大いに有効でない

```
階級 0 ( 1 ):  | ?
階級 1 ( 12 ): | ************
階級 2 ( 56 ): | ************************************************************
階級 3 ( 30 ): | ******************************
階級 4 ( 2 ):  | **
階級 5 ( 0 ):  |
```

⑦与えられた問題を解くためのプログラムを作成する場合、何が障害となっていますか？（多肢選択可）
設問 7 （有効延べ度数を100 に換算）
1．与えられた問題そのものの意味が分からない
2．問題解決のための手順（アルゴリズム）が分からない
3．プログラムをどのように構造化したらよいか分からない
4．アルゴリズムの表現としての流れ図が分からない
5．表現されたアルゴリズム（たとえば流れ図）に従ってコーディング出来ない
6．プログラミング言語（コボル、フォートラン、ベーシック等）の文法が分からない

```
階級 0 ( 1 ): | ?
階級 1 ( 9 ): | *********
階級 2 ( 23 ):| ************************
階級 3 ( 36 ):| ************************************
階級 4 ( 7 ): | *******
階級 5 ( 14 ):| **************
階級 6 ( 11 ):| ***********
```

（3）アルゴリズムとプログラミング言語について

⑧アルゴリズムの理解のし易さは、使用している言語によると思いますか？
設問 8 （有効度数を100 に換算）
1．大いによる 2．よると思う 3．どちらとも言えない 4．よらないと思う 5．大いによらない

```
階級 0 ( 1 ):  | ?
階級 1 ( 2 ):  | **
階級 2 ( 45 ): | *********************************************
階級 3 ( 48 ): | ************************************************
階級 4 ( 4 ):  | ****
階級 5 ( 1 ):  | *
```

⑨アルゴリズムをコーディングする際の容易さは、使用している言語によると思いますか？
設問 9 （有効度数を100 に換算）
1．大いによる 2．よると思う 3．どちらとも言えない 4．よらないと思う 5．大いによらない

```
階級 0 ( 2 ):  | ??
階級 1 ( 2 ):  | **
階級 2 ( 45 ): | *********************************************
階級 3 ( 47 ): | ***********************************************
階級 4 ( 6 ):  | ******
階級 5 ( 0 ):  |
```

```
⑩あなたにとって、アルゴリズムの表現としては、どの言語が一番分かり易いですか？
設問 10 (有効度数を100 に換算)
  1．コボル  2．フォートラン  3．ベーシック  4．パスカル  5．C  6．その他

階級 0 ( 2 ): | ??
階級 1 ( 42 ): | ******************************************
階級 2 ( 24 ): | ************************
階級 3 ( 30 ): | ******************************
階級 4 ( 3 ): | ***
階級 5 ( 0 ): |
階級 6 ( 1 ): | *

⑪あなたは、どの言語が一番好きですか？
設問 11 (有効度数を100 に換算)
  1．コボル  2．フォートラン  3．ベーシック  4．パスカル  5．C  6．その他

階級 0 ( 1 ): | ?
階級 1 ( 36 ): | ************************************
階級 2 ( 22 ): | **********************
階級 3 ( 34 ): | **********************************
階級 4 ( 6 ): | ******
階級 5 ( 0 ): |
階級 6 ( 2 ): | **

⑫ベーシック、フォートラン、コボルとプログラミング言語を学んで来て、だんだんプログラミングが分かって来た。
設問 12 (有効度数を100 に換算)
  1．大いに賛成  2．賛成  3．どちらとも言えない  4．反対  5．大いに反対

階級 0 ( 0 ): |
階級 1 ( 2 ): | **
階級 2 ( 33 ): | *********************************
階級 3 ( 52 ): | ****************************************************
階級 4 ( 11 ): | ***********
階級 5 ( 2 ): | **

⑬ベーシック、フォートラン、コボルとプログラミング言語を学んで来て、プログラミングが好きになって来た。
設問 13 (有効度数を100 に換算)
  1．大いに賛成  2．賛成  3．どちらとも言えない  4．反対  5．大いに反対

階級 0 ( 0 ): |
階級 1 ( 2 ): | **
階級 2 ( 22 ): | **********************
階級 3 ( 57 ): | *********************************************************
階級 4 ( 15 ): | ***************
階級 5 ( 4 ): | ****
```

択）としては，40%の学生が「段落（或いは，手続き）をそれぞれ独立に書く事が出来る」，28%が「問題を大きな部分から小さな部分に分割してプログラム出来る」，20%が「部品を用意してそれを組み合わせてプログラムを構成出来る」と回答している．

(2)アルゴリズムとプログラミングについて

⑤「プログラムを作る際にヒントがないとプログラム出来ない」に対して，「大いに賛成」と「賛成」を合わせると，87%の学生が何らかのヒントがないと解答出来ないと答えている．⑥「プログラミングをする場合に流れ図は有効だと思いますか？」に対しては，「大いに賛成」と「賛成」を合わせて，68%の学生が「有効である」と答えている．⑦プログラムを組む際の障害（多肢選択）については，第1位が，「プログラムをどのように構造化したら良いか分からない」(60%)，第2位が，「問題解決のための手順（アルゴリズム）が分からない」(38%)，第3位が，「表現されたアルゴリズム(例えば「流れ図」)に従ってコーディング出来ない」(24%) となっている．

(3)アルゴリズムとプログラム言語について

⑧「アルゴリズムの理解のし易さは使用している言語によると思いますか？」に対しては，47%の学生が「大いによる」又は「よると思う」と答え，48%の学生が「どちらとも言えない」と答えている．⑨「アルゴリズムをコーディングする際の容易さは，使用している言語によると思いますか？」に対する回答の様子も⑧とほぼ同様で，「大いによる」又は「よると思う」と答えた学生は，47%で，「どちらとも言えない」と答えた学生は，48%であった．⑩では，「アルゴリズムの表現とし

てはどの言語が一番分かり易いか？」を尋ねているが，コボル(42%)，ベーシック(30%)，フォートラン(24%)であった．最後に，「ベーシック，フォートラン，コボルと3つの言語を学んで来て，だんだんプログラムが分かってきたか」に対しては，52%の学生が「どちらとも言えない」，35%の学生が，「大いに賛成」又は「賛成」，13%の学生が，「大いに反対」又は「反対」であった．更に，「ベーシック，フォートラン，コボルと3つの言語を学んで来て，プログラミングが好きになって来たか」に対して，57%の学生が「どちらとも言えない」，24%の学生が，「大いに賛成」又は「賛成」，そして19%の学生が，「大いに反対」又は「反対」であった．

(5) 評価と議論

i）アルゴリズム理解度調査

基本制御構造については，(A)「流れ図」→「コーディング」，(B)「コーディング」→「流れ図」とも全体としては7割前後の正答率に達している．学生にとっては，(B)「コーディング」→「流れ図」の変換の方が難しいようである．4つの基本制御構造の中では，③反復構造が，全てのクラスに於いて正答率が極端に低い．反復構造の「流れ図」の図形的認識（「流れ図」のパターン全体を一つのまとまった処理として視覚的に把握する）が不十分なことを示している．複合制御構造についての理解度はかなり低い．複合制御構造の構成要素には，反復構造が組込まれているから，複合制御構造の正答率が，基本制御構造の中で最も低い正答率を更に下回ったものになるのは理論的に理解できる．ここに於いても基本制御構造の正確な理解が要求される．

ii）学生の自己申告による「プログラミングに関する質問紙調査」

(1)構造化プログラミングについて

構造化プログラミングについての理解はまだ3割程度の学生に止まっている．又，「構造化プログラミング」が分かり易いかどうかについても学生の評価は余り芳しくない．これは，勿論構造化プログラミングの理解度の低さに直接的理由があるが，授業時に於ける構造化プログラミングの提示と説明の仕方に不徹底なところがある事にも依っていると思われる．しかし，構造化プログラミングが本来的に持っている筈の「分かり易さ」については，ある程度評価している．更に，その「分かり易さ」の理由としては，4割の学生が構造化プログラミングの重要な特徴である，「段落（或いは，手続き）をそれぞれ独立に書く事が出来る」を指摘している．興味深いことに，構造化プログラミングの2つの手法である「下降型（トップダウン）」については，3割の学生が，「上昇型（ボトムアップ）」については，2割の学生が，構造化プログラミングの「分かり易さ」の理由として指摘している．

(2)アルゴリズムとプログラミングについて

「プログラムを作成する際にヒントがないとプログラムが書けない」という学生がほぼ9割に達するという学生の申告は大いに考えなければならない．プログラムを書く際に障害であると感じている要素を見てみると，6割の学生が「プログラムの構造化の困難さ」を挙げている．これには，コーディングの構造単位である制御構造の理解度不足も関係しているし，又プログラムをどのように段落，或いは，モジュールに分けるべきかという戸惑いも含まれているように思われる．3割の学生が指摘している「問題解決のための手順（アルゴリズム）が分からない」という評価に対しては，標準的な定型処理や代表的なデータ処理のアルゴリズムの型（タイプ）を理解させ，例題を多くこなさなければならないだろう．「表現されたアルゴリズム(例えば「流れ図」）に従ってコーディング出来ない」という学生については，基本制御構造及び複合制御構造についての「流れ図」と「コーディ

ング」の間の書き直しについての基礎的な理解と実習の徹底から始める必要があろう．7割の学生が「流れ図」の有効性を認めていることは心強い事である．

(3)アルゴリズムとプログラム言語について

「アルゴリズムの理解のし易さが，使用している言語による」と考えている学生と「よらない」と考えている学生の割合はほぼ同じである．又，「アルゴリズムをコーディングする際の容易さが，使用している言語による」と考えている学生と「よらない」と考えている学生の割合もほぼ同じである．これは，論理の問題としてアルゴリズムを考えれば，個々の具体的な言語によらないと言えるし，又，実際のコーディングを考えると，個々の言語に依存する所があるのは当然と言えるし，もっともな評価結果と言えるかも知れない．アルゴリズムの表現として，コボル(42%)が一番分かり易いという感想は，意識的にコボルの「構造化プログラミング言語処理系」を重視している「プログラミング論 III」の学生の評価としては，当然の結果であろう．⑪「どの言語が一番好きか」という問いに対して，コボル (36%)という結果はやや意外であった．というのは，コボルは他の言語に比べて，データ部の定義が重いので学生がコボルを敬遠するかとやや危惧していたからである．⑫，⑬の設問については，2年間の「プログラミング論 I，II，III」の履修を終えた時点で，「だんだんプログラムが分かって来た」及び「プログラミングが好きになって来た」が，逆の自己評価を相対的に上回っていると言うことは，大いに勇気づけられるところである．今後のプログラミング教育の成果に期待したい．

6．結　論

この小論では，プログラミング言語教育の立場からアルゴリズム教育を考えた．構造化プログラミングとデータ構造に関する理解を基本に据えて，基本制御構造と複合制御構造とから構築される「アルゴリズムの構造化」を「流れ図」という表現形式を用いて視覚化し，構造化プログラミングの手法に基ずいてコーディングする方法論の具体的展開を試みた．北星学園大学経済学部経営情報学科に於ける基礎的情報処理教育も緒についたばかりである．ここで論じられた方法の具体的実践の継続と評価の中から，アルゴリズム教育のより好ましいあり方が検討されて行くことを期待したい．

謝　辞

この小論を纏めるに当たって，次の方々に御教示頂いたことを感謝致します．
田中　二郎氏（札幌学院大学社会情報学部），奥村　真司氏（北海道工業大学工学部経営工学科），片山　敏之氏(北星学園大学経済学部経営情報学科)．又，今回の「情報処理教育研究会」の世話人の方々に御無理をお願いし，いろいろ御配慮頂いた事に対してお礼申上げます．

文　献

(1) 東和コンピュータマネジメント著：情報システムの開発と設計，啓学出版，東京(1987)．