ビジュアル環境を活用したプログラミング教育の試み

―― Delphi（Object PASCAL）の場合 ――

森田 彦・石川 高行・高橋 哲男

マイクロソフト社の Visual Basic や Visual C++ 等の普及に伴い，現在，パーソナルコンピュータ上でのプログラム開発環境としては，いわゆるビジュアル・プログラミング環境が標準になりつつある．これは，Windows 上でのアプリケーション開発の効率性を高めるという利点がある一方，初学者にとってみれば，プログラミング言語の文法事項等と共にビジュアル・プログラミングを実現するライブラリの用法も学ぶ必要があるため，従来のプログラミング言語学習に比べて，学習負担が大きくなるという側面がある．本稿は，この問題に対する一つの答えとして，代表的なビジュアル・プログラミング環境を提供しているインプライズ社の Delphi（Object PASCAL）についての，プログラミング教育の一案を提唱している．なお，ここで提示された講義案は 2000 年度の本社会情報学部のプログラミング科目で実施する予定である．

1．はじめに

本社会情報学部では，主に 2 年次の学生を対象としてプログラミング言語 A（C 言語「BorlandC++ Builder」）およびプログラミング言語 B（PASCAL 言語「Delphi」）が開講され，学生は，これらの内，少なくともいずれか一つを単位取得する事を義務づけられている．その内，ここで具体的に採り上げるのはプログラミング B, すなわち Delphi に関する教育である．この科目は，週 1 回 2 コマ連続で行う通年科目であり，実習室におけるプログラミング演習が主体となるものの，適宜講義を行い文法事項やプログラムの論理構造等の解説を行っている．この様に，本科目の実施形態は演習が主であるが，以下では，便宜上〝講義〟という呼称で引用する事にする．

ここで，本学部におけるこれまでのプログラミング教育の経緯についてふれておく．上記のように C 言語と PASCAL 言語を採り上げたのは 1997 年度からであり，それ以前は FORTRAN および COBOL をプログラミング言語として用いていた．その際，森田が担当していたのは FORTRAN であり，そこでは教育の目的を，特定のプログラミング言語に習熟することではなく，アルゴリズムを考える論理的思考能力を培う事に置いた．この教育目標を実現するために，その間，アルゴリズムを考える能力に関する考察（森田・新國・原田，1993）や，プログラミング習熟度の高い学生の特性の分析等を行い（森田，1995），それを教育内容および学生の指導に反映させて来た．そしてこれは相応の効果を上げたものと判断している．

Morita Hiko 札幌学院大学社会情報学部
Ishikawa Takayuki 北海道大学大学院教育学研究科
Takahashi Tetsuo 北海道大学大学院教育学研究科

ところが，1997年度にいわゆるビジュアル・プログラミング環境[1]をベースにしたDelphiを導入した際に，次のような困難に直面した．プログラミングを行うためには，使用するプログラミング言語の文法事項と，「条件判断」や「繰り返し」等の基礎的な論理構造を理解しなければならないが，ビジュアル・プログラミングの場合はこれに加えて，ビジュアル・コンポーネント・ライブラリ(VCL)[2]の用法を学習しなければならない．そのため，プログラミング経験のない初学者にとっては，従来の言語，例えばFORTRANの学習に比べて学習の負荷が高くなってしまう．それに対処するために我々が採ってきた教育手順は次の通りである．まず，最初はVCLを用いず，入出力をコンソールに限定した従来型のプログラミング環境で，言語の文法事項および基礎的な論理構造を理解する学習を行う．これは，最初はVCLにふれずに文法事項等の学習のみに集中した方が，学生の混乱を防ぐ事ができ，したがって学習効果も上がると考えたからである．そして一通り，基礎的な事項の学習を終えた後にVCLを用いたビジュアル・プログラミングの開発形態を学んで行く様にした．ところが，この方法では，VCLを活用したプログラミング・スタイルが，それまでの（コンソールを入出力とした）プログラミングスタイルと大きく異なるため，多くの学生がそのギャップに戸惑い[3]，ビジュアル・プログラミングを身につけるまでかなりの時間を要するという事態に陥ってしまった．

これに対処するのに，全くVCLを用いずに従来型のプログラミング教育を一貫して行う，というやり方も考えられる．アルゴリズムとデータ構造の理解に特化した教育等，目的に応じてはその様なやり方も有効であろう．しかし，一般的に言えば，ビジュアルなプログラミング開発環境が標準になりつつある現在，それにふれない事のデメリットは大きいと言わざるを得ない．また，ビジュアル環境では，学生がそれまで慣れ親しんだWindowsアプリケーションの動作形態，例えばボタンクリックやメニュー選択による処理の実行等を，比較的簡単に実現できるために，学生の興味・関心を引きやすくまた達成感も与えやすいという教育的効果も一方ではある．そこで我々は，VCLはできる限り活用するという方針をとる事にした．

それでは，VCLを活用しつつも，初学者にとって負荷の少ない，つまりハードルの低いプログラミング教育は実現できるのであろうか．本稿では，この問題に答える一つの講義案を提唱する．そこでは，VCLの使用を前提とし，従来型のコンソールを使用したプログラム解説は一切行わない．一般には，基礎的な文法事項を学ぶ事と，VCLの使用法とが混在するとその分だけ（学習する側の）負担が増えるという点が危惧されるが，本案はそれに対する一つの答えを提示している．現時点ではまだ構想段階であるが，半期分(半年分)の講義案として用意したその全体像を第2章にまとめる．

一方，最近の学生の動向とも関連して，従来に比べてプログラミングの習熟度の差が大きくなってきた，という問題がある．この傾向に対応するためには，各々の習熟度に応じて学習を進めることができる自学自習形式の要素を取り入れる事が有効であろう．そこで，講義内容（教材）作成に当たっては，自学自習形式で学ぶことができる様配慮する事とした．また，習熟度に大きな差が出るのは，基礎事項の理解に一つでもつまずくと以降の理解に大きな支障が出てしまう，というプログラミング学習の特性にも起因していると考えられる．そこで，できるだけ理解しやすい形で，あるいは別の言葉で言うと，理解する上でつまずく事の少ない形でシステマティックに教育を行う事が（これまで以上に）必要であると言える．そのポイントを，我々は新し

い概念を学習する際の提示の仕方にあるのでは，と考えた．これまでプログラミング教育では，「単純なもの（あるいは特殊なもの）」から「複雑なもの（あるいは一般的なもの）」へ，という順序で様々な概念を学習する事が普通であった．確かに自然な流れではあるが，これまでの我々の経験では，一般化する過程で学生がそれを受け入れる必然性を感じる事ができず，なかなか一般化した形式を身につける事ができないという場面に少なからず出くわした．場合によっては，「一般から特殊へ」という提示の仕方の方が概念を受け入れやすいのではないだろうか．そこで，我々は理解しやすさのポイントをこの点に置きながら，具体的な学習内容の作成に当たった．その考え方の詳細と具体例を第3章に記す．

本稿で提示した講義案は2000年度実際にプログラミングBで実施する予定である．そこで，今後の計画について，最後の第4章で簡潔に述べる．

2．講義案の全体像

この講義案の最大の特徴は，従来のようなコンソールプログラムを一切扱わずにGUI (Graphical User Interface) プログラムの作り方を教える点にあり，それを通じてDelphiというオブジェクト指向言語の考え方の一端に触れることを目的とする．GUIプログラムを前面に押し出したのは，それがコンソールプログラムに比べて学生の興味を大いにひくばかりでなく，なるべく最新の形態にふれる事が，教育学的にも望ましいと考えられるからである．

講義は，重要な学習項目であるオブジェクトから始まり，オブジェクトが〝退化〟したものとして変数をとらえ，以下「イベント」，「演算子」，「変数の型」そして「制御命令」など，プログラミングの基礎となる内容について一通り学んで行き，最終的に半期で「関数」と「ソースファイルの分割」まで到達する事を目標とする．ここまで来ると，学生は，プログラムを複数のモジュール(あるいは関数)の集合として捉える考え方を身に付けることができる．これは，少し本格的なプログラムを作る際の基本的な考え方であり，またオブジェクト指向の考え方を理解するための重要なステップであると言える．この観点からみて，上で述べた到達目標は，導入時のプログラミング教育としては妥当であると考えている．

講義案の，具体的な単元名とその教育内容は表1にまとめられている．以下，教育内容を関連するまとまり毎に解説していく．

2.1 オブジェクト，変数

Delphiでは，プログラムの操作対象はメモリ上のデータ，すなわちオブジェクトである．プリンタで文字を印刷するなど，一見周辺機器を操作しているように見えるプログラムにおいても，Delphiの上では周辺機器を直接操作している訳ではない．ただ，印刷したい内容を持つオブジェクトに「印刷せよ」と命令しているだけである．このように，オブジェクトはあらゆるDelphiプログラムの基本であるから，まず何よりも先にオブジェクトが教えられなくてはならない．ビジュアル・プログラミングで用いられるVCLの各コンポーネントもオブジェクトである．

VCLの一つで，図1に示す様な，文字入力欄としての機能を持つTEditコントロール[4]（オブジェクトの一つ）を用いると，3.1で説明する様に，コーディングを一切する事なく，オブジェクトの概念を，ひいては変数の概念を獲得し，さらにその値を変更してみることが可能になる．

一方，Delphiが世に現れるまでは（PASCAL言語の場合），変数について学習するためには，まず

```
var
  a: Integer;
```

表1 講義案目次

単 元 名	教 育 内 容
初めてのプログラムを作ってみよう	(何もない雛型の作成)
演習で使用するメニュー	(演習で用いるメニューを限定することによって混乱を防ぐ)
入力欄を置こう	(オブジェクトの典型としてのTEditコントロール=入力欄)
プロパティを変えてみよう	(プロパティの変更,オブジェクトは値を保持するものであること)
プログラムを書こう	(イベント, コードの書き方)
コンポーネントに名前をつけよう	(識別子の名前は自由に決められること)
〝さん〟づけしよう	(文字列の足し算)
場合分けしよう	(if文)
入力欄の内容の入れ替え	(画面に表示しなくてもよい入力欄)
変数	(入力欄の退化としての整数型変数の宣言)
足し算プログラム	(整数の足し算)
変数の型	(変数の型)
繰り返し/for文	(for文)
for文を作るコツ	(for文をΣ=1+2+3+…のように展開)
繰り返し/while文	(while文)
複数行入力	(配列の具体物としてのTMemoコントロール=複数行入力欄)
行数を知る	(行数がTMemo.Countで表されることを知り,繰り返し処理の引数とする)
配列変数	(配列変数)
絶対値関数abs()	(関数の使い方)
関数の使い方/uses節	(標準ではないライブラリ関数のuses節宣言)
関数の使い方/変数の型	(関数とやり取りする変数の型)
下請け会社	(関数のブラックボックス化, 変数のスコープ)
ShowMessage	(関数の退化としての手続き)
関数を作ろう	(関数の自作)
関数の利点	(構造化プログラミングの利点)
関数の順序と宣言	(関数の宣言)
プログラムの分割	(ソースファイルのユニット分割)

等と，変数宣言をコーディングしなければならなかった．さらに，その値を表示させるためには，「write」あるいは「writeln」等の出力文について触れざるを得ず，必然的に，変数概念の獲得とプログラミング（コーディング）とは同時に行なわれる必要があった．

この点を考えると，変数はオブジェクトの〝退化〟したもの，あるいは，変数はオブジェクトという一般概念を〝特殊化〟したもの，と捉える教育方式は，学習者の学習負担を減ずるという観点から有効であると言えよう．その点については，3.1で詳しく述べる．

2.2 イベント

かつてMS-DOSの時代は，「プログラムが起動された時に何を行なうか」ということをコーディングしていった．しかし，MS-Windowsの時代になると「イベント・ドリブン（駆動）」という考えが持ち込まれ，「イベント（マウスボタンがクリックされることなど）が発生した時に何を行なうか」を記述することがコーディングの主な目的となった．このように，人間がプログラムに処理開始を命令するものとして，イベントは重要な概念である．本講義案では，イベントを発生させる典型的なコントロールとして，図2に示す

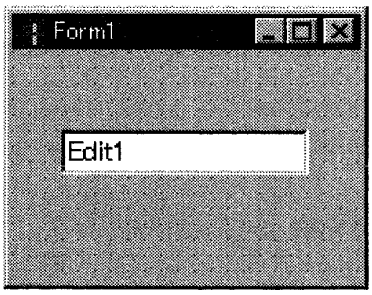


図1 入力欄（TEdit コントロール）

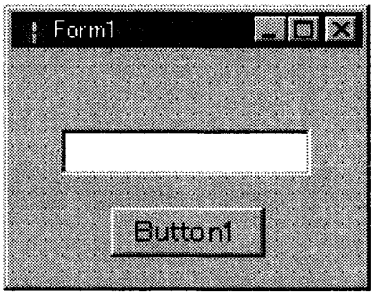


図2 TButton コントロール

TButton コントロールを用いる．TButton コントロールはまさしく押されるため（イベントを発生させるため）に存在するものだからである．

2.3 演算子，変数の型

例えば，「＋」という演算子は，それぞれの型毎に処理方法が異なる．整数を「12＋34」と演算した結果（46）と文字列を「'12'＋'34'」と演算した結果（'1234'）は異なる．つまり，演算子を教えるということは，変数の型を学生に意識させることでもある．この観点から，ここでは，両者を同時に取り扱っている．

2.4 制御命令

if，for，while などの，「条件判断」や「繰り返し」命令は，プログラムの流れを制御するので制御命令と呼ぶ．ここまでがプログラミング言語指導の中でも本当に基礎の部分であり，この段階まで学んだ学生は単一モジュールからなる単純なプログラムを作ることができる．

なお，制御命令を教える際は，3.2 で具体的に説明する様に，字下げを強調することによって，プログラムの論理構造を学生に意識させる様にする．

2.5 関数，手続き

大きな，あるいは本格的なプログラムは，多くのモジュール（関数）の集合である．モジュールとは，それ自身が小さなプログラムであり，ある決まった機能を持つ（関数までしか学んでいない段階では，１つ１つの関数がモジュールであると考えて良い）．例えば，３次元アニメーションプログラムを作る場合を考えよう．その場合，３次元物体を定義したり動かしたりするモジュール，３次元物体を画面に表示するために２次元平面に射影変換するためのモジュール，実際に画面表示を担当するモジュールなどに分けてプログラミングする．こうすることによってプログラミングの生産性が向上し，保守性が上がり，またモジュール単位で他のプログラムでも再利用できるようになる．このように，本格的なプログラムを作る事が出来る様になるためには，関数は避けて通れない．そのため，ここでの関数の指導は，Delphi（Pascal）に予め用意されている組み込み関数の利用よりも，関数を自作することに重点を置く．なお，関数はそれ自身が小さなプログラムであるため，関数を学ぶ前に制御命令までの基礎を終えておく必要がある事は言うまでもない．

本講義案では，手続きは関数が退化したもの（返り値を持たない関数）であると考える．そのため，市販の教科書とは逆に，手続きより先に関数を教える．

2.6 ソースファイルの分割

本格的なプログラミングでは，実現したい機能をなるべく独立性が高いモジュールに分割して実装する．そして，１つのユニット（ソースファイル）には１つのモジュールが収まっているのが普通である．例えば，先ほどの３次元アニメーションプログラムの例において，それぞれのモジュールを別々のユニットに収めておいたとしよう．すると，例えば，

魚眼レンズで見たようなアニメーションを作りたい場合には，魚眼型射影変換ユニットを作って従来の射影変換ユニットと交換するだけで良い．同様に，画面表示を高速化するためソリッド（中身が詰まった物体）ではなくワイヤフレーム（針金で作った枠）で表示したいのなら，ワイヤフレームユニットを作り従来の表示ユニットと交換するだけで良い．このように，プログラムが本格的になるほどソースファイル分割は重要になる．

また，ソースファイルを複数のユニットに分割する際に，関連性が高い変数や関数を1つのユニットに集めると，オブジェクト指向プログラミングで重要な概念であるクラスの原始的な形態となる．これは，クラスという概念の端緒を学生に触れさせることとなり，後に本格的にクラスを学習する際の足がかりとなるであろう．

3．講義案の特徴的な点とそのねらい

本講義案の特徴を一言で述べるならば，「一般から特殊へ」という言葉が最もふさわしい．

従来のプログラミング教育の方法は，「拡張方式」と名づけることができよう．拡張方式では，なんらかのプログラムを完成した後，「このプログラムに○○の機能をつけ加えてみよう」と述べて，新しい機能を実現するための言語文法を説明する．あるいは，複雑になった命令群を見やすくするために「実はこういう書き方があるのだ」と述べて，繰り返し文や手続き化・関数化などの新しい文法を教える場合もある．

拡張方式は，学習者の動機づけが希薄になるという弱点をもつ．実際，次のような場面によく出くわす．

教員：「こういう機能を加えてみよう」．
学生：「このままでもできるから，これでいい」．
教員：「命令を短く簡単にする方法がある」．
学生：「別にこのままでも構わない」．

ここに，教える側にとって便利な機能や文法が，必ずしも学習者にとっても便利であるとは限らないことに注意しなければならない．例えば，for や while などを用いた「繰り返し文」を教える際には，自然数の和を求める問題が例としてよく用いられる．1から相当大きなある数までを1つずつ加えることの大変さを伝えた上で，「繰り返し文」の〝ありがたみ〟を教えるという流れをとる．しかし，学習者は1つずつ加える方法を好みやすい．そうすることでいずれは〝確実に〟答が求められることを知っているからである．「繰り返し文」は，学習者にとって，覚えることを増やす〝ありがたくない〟ものになるのである．

本講義案では，上のような拡張方式ではなく，可能な限り，最初から拡張した形を教えようとする．教える側が便利であると確信し是非教えたいと考える文法内容ならば，拡張しつつ徐々に教えるのではなく，最初に（全ての要素が入った）一般形を提示しようというのである．つまり，「一般から特殊へ」，あるいは，「重要なものを先に，重要でないものを後に」という原則である[5]．

一般化あるいは抽象化といったことは，重要であることは認識されつつも，〝難しい〟と考えられがちである．そのため，一般化に先だって数多くの個別な事例を提示することは，様々な場面で〝有効な〟教育方法として登場する．しかし，中には，学習者の認識能力を軽んじて，最初から一般化した形で教えることをあきらめてしまってはいなかっただろうか．特にプログラミング教育においては，拡張方式が長い間幅を利かせてきた様に思われる．そして，その方式は，学習者にとって，次から次へと覚えることを増やさせ，何が重要で何が些末なのかを判断するのを困難にさせるような，(学習側にとって)好ましくない教育方法になってきた可能性はないだろうか．

このような認識から，我々は，可能な限り

最初から拡張した形を教えようとするのである．ただし，〝可能な限り〞という条件を忘れてはならない．自然数は正の有理数（分数）の特殊ケースであるが，だからといって，小学校の数指導で有理数を先に教えることは不可能である．当然の事ながら，その有効性は適用範囲があり，我々はその点を認識している．

以下では，「一般から特殊へ」の原則を適用でき，これまでの拡張方式とは異なる順序で教えることが可能であると考えられる教育内容の具体例として，3.1でオブジェクト（変数），3.2でif文について，それぞれの指導方法を解説する．その中で，「一般から特殊へ」の原則の有効性が明らかになるであろう．

3.1 オブジェクト（変数）

本講義案では，オブジェクトは変数の一般形であるとの立場から，オブジェクトとして入力欄，すなわちTEditコントロール[4]を教え，後にそれが退化したものとして変数を導入する．以下，3.1.1では，最初に入力欄（TEditコントロール）を採り上げる事の経緯について述べ，3.1.2において，入力欄を用いて変数概念の獲得へと導く道筋を示す．

3.1.1 最初に入力欄（TEditコントロール）を教えること

コンピュータは中央演算処理装置（CPU），主記憶装置（メモリ），補助記憶装置（HDなど），周辺機器（device）からなり，CPUが直接処理するデータはメモリ上に置かれる．機械語はメモリ上のデータを生のまま扱うが，それでは不便なので，高級言語ではメモリ上のデータの意味あるひとまとまりに識別子（identifier）をつけ，その識別子によってデータを操作している．これが変数である．Delphiのようなオブジェクト指向言語では，変数（やその集合体＝レコード型変数）にメソッド（処理方法，メンバ関数とも言う）を関連付け，それをオブジェクトと呼んでいる．2.1で述べたように，オブジェクトはあらゆるDelphiプログラムの操作対象である．「何を操作（処理）するのか」が分からなければプログラムは書けないのだから，第一にオブジェクトを教えるべきであると考える．

ところで，オブジェクトは複数の変数から成り立っているので，実は，オブジェクトの概念を教えることは変数の概念を教えることを包含している．何らかの値を保持するものである，という点でオブジェクトと変数は同質のものなのである．この点が，〝変数への退化〞へ至る伏線になっている．

少し具体的にみてみよう．オブジェクトとして，2.1でふれた通り，入力欄としての機能を持つ，TEditコントロールを最初に導入する．これはText，Height，Width，Color等という変数（Delphiではプロパティと呼ばれる）から成り立っており，その内Textプロパティが，入力した文字列を保持している．そこで，このオブジェクトが退化し，Textプロパティのみを保持している状態になったものとして変数を導入する事ができるのである．その説明の詳細は，次の3.1.2に示す．

さて，それでは，数あるオブジェクトの中からなぜ入力欄（TEditコントロール）を用いたのか，その理由を以下に説明しよう．

第一に，学生の興味・関心をひくGUIというビジュアルな環境を最大限活用する事が有効である，と考えたからである[6]．GUIプログラムは，入力欄やボタンなどの，プログラム実行時に画面上に表示されるコントロールを用いて作る．全ての学生は1年次に「情報処理」という科目で既にMS-Windowsの使い方を学んでおり，TEdit（文字入力欄）のようなコントロールは，既に何度も経験済みで，それに慣れ親しんでいる．

次に，数あるコントロールの中でも入力欄が箱としてのイメージを持っていることが，第二の理由である．ここで，オブジェクトではなく変数を最初に教えようとする市販の教科書を再び見てみよう．もともと変数はメモ

リ上の存在であり，その値を表示させる命令がなくては目で見ることはできない．だから，変数の存在をイメージさせるため，「服の保管にはタンスがあり，本の保管には本棚があるように，プログラム中で使用する数値や文字列，あるいはオブジェクトなどを保管するためには，箱を用意する必要があります．この箱のことを変数と呼びます．」（三原，1999：65）というように，変数はよく箱にたとえられる．箱にたとえるのなら，最初から箱のような外見を持つ入力欄を用いた方が，よりリアルにイメージをつかみやすい．

第三点として，入力欄の代表的なプロパティであるTextプロパティの値が常に表示されるという利点が挙げられる．従来のコンソール環境でのPASCALや，GUI環境を生かせていない市販の教科書では，変数の値を画面に表示するには随時write文などを用いる必要がある．しかし，変数の内容を知ろうとするたびにwrite文などが必要になることは煩雑である．入力欄を用いれば，（Textプロパティの）値は常に表示されており，このような煩雑さがない．

最後に，入力欄は実行中にも手作業でTextプロパティの値を変更できる．これは，プログラムが正しく動いているかどうかを確認したい場合に大変便利である．その点，Textプロパティを持たないTButtonコントロールなどの場合，代わりにCaptionプロパティがボタン表面に表示されているが，実行中に手作業で値を変更できないので，後に変数に退化させるには不向きである．

以上で，変数への退化を学習する素材としての，入力欄の有効性が示された．

3.1.2 入力欄から変数への退化

この単元では，2つの入力欄の値を入れ替える，という処理を考える．まず，これを実現するためにはもう1つ別の中継用入力欄が必要であることを理解させ，第3の入力欄（EditTemp）を作らせる．この中継用入力欄が，データの保管場所としての変数に，これから〝退化〟して行くのである．さて，この中継用入力欄は，入力文字，つまりデータの一時〝保管場所〟としてのみ，必要とされるものである．したがって，画面上にそれが表示される必要はないので，次に，Visibleプロパティをfalseにする．こうすることによって，中継欄は画面に表示されなくなる．これが第1段階の退化である．（Web上の教材として用意した）ここまでの解説を，図3に示す．

最後に，中継欄に必要な機能はそこに入力された文字列の値を保持することだけであること，すなわちTextプロパティだけである事を確認する．したがって，例えばその他のHeightやWidth, Colorプロパティ等にはどんな値を入れても結果に影響しない，つまり，意味がない事を示す．そこで，必要充分な機能を持つもの，つまりTextプロパティのみを有するものとして変数を導入する．具体的には，これは文字型の変数である．これが第2段階の退化である．この部分の説明は図4に示されている．

以上の説明で，入力欄（TEditコントロール）の特殊なものとして変数を導入し，その概念を理解させる事が可能であろう．

3.2 if文

一般にif文は，

(a) 条件とよばれる命題

(b) 命題が真の時の実行文

(c) （必要ならば）命題が偽のときの実行文

という3つの要素からなる構造をもつ．Delphi（Pascal言語）では，if, then, elseが，(a)(b)(c)それぞれの標識になる．まず，典型的なif文指導の例として，学生向けの参考書にもなっている『Delphiで作るWindowsプログラム』（藤本，1996）を見てみよう．このテキストにおけるif文指導の順序は，図5のようになっている．すなわち，

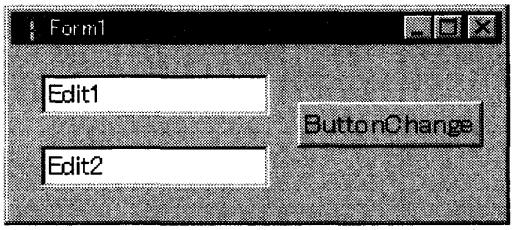
(1) ifとthenのみをともなうもの

目次に戻る
前のページへ
次のページへ

入力欄の内容の入れ替え

新しいプログラムを作ります。

まず、次のようにフォームを作ってください。



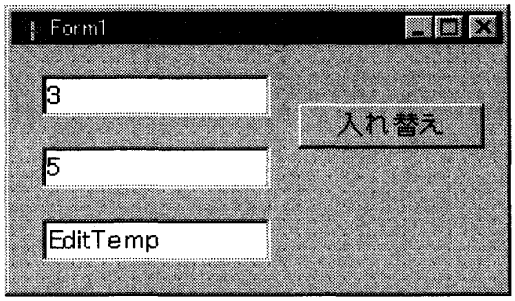
ボタンの表示は「入れ替え」にし、Edit1のTextは「3」、Edit2のTextは「5」にしてください。

ボタンを押すと上の欄と下の欄が入れ替わる、というプログラムを作るにはどうしたらいいでしょうか。例えば

```
procedure TForm1.ButtonChangeClick(Sender: TObject);
begin
  Edit2.Text := Edit1.Text;
  Edit1.Text := Edit2.Text;
end;
```

というプログラムでは、1行目が実行された時点でEdit2の内容が消えてしまいます。(試しに実行してみてください。)だからといって1行目と2行目を入れ替えてもうまくいきません。

この場合、Edit2の内容を取り敢えず保存しておく入力欄が必要です。次のように、データの中継用の入力欄を加えてください。(Temp は temporary=一時的 の略です。)



この中継用入力欄は空欄にし、プログラムを次のように変えてください。

```
procedure TForm1.ButtonChangeClick(Sender: TObject);
begin
  EditTemp.Text := Edit1.Text;
  Edit1.Text := Edit2.Text;
  Edit2.Text := EditTemp.Text;
end;
```

どうですか。3と5が入れ替わりましたか。

このままでもプログラムは正しく動きますが、中継用入力欄が画面に表示されるのは格好悪いですね。何とかして中継用入力欄を画面から隠しましょう。

オブジェクトインスペクタで、EditTempのVisibleというプロパティ(属性)を見てください。True(真)になっていますね。このVisibleとは、(そのコンポーネントが)見えるか見えないかを決めるプロパティ(属性)です。これをfalse(偽)にして実行してみましょう。(Visibleの右の欄をクリックすると、右端に　というボタンが現れます。このボタンを押してfalseを選びます。) これで、中継用入力欄が画面から消え、一応まともな入れ替えプログラムになりました。

図3 「入力欄の内容の入れ替え」

(2) elseをもともなうもの
(3) 処理する文が複数になる，すなわち「複合文」の場合．「begin～end」の形で囲むことが必要になる

という指導順序になっている．ここでは，(1)を〝基本的な〟if文として，それに命題が偽の時の処理を加えて拡張したものが(2)であり，さらに，処理する文が複数になった場合にも対応できるように文法を拡張して(3)を指導している．これが，典型的な拡張方式によるif文の指導である．

しかし，if文を教えるとはどういうことかを，改めて考えてみよう．if文を教えるとは，if文を〝完全に〟教えることを意味する．ここに，if文を〝完全に〟教えるとは，条件命題の標識であるifを用いたあらゆる構造の文を自由に作れるようにすることである．このようなif文指導の目標に照らして，図5の指導順序は適切だといえるのだろうか．

あらゆる構造のif文を作るための要素をすべて含んでいるのは，(3)である．したがって，我々の構想する講義案では，(3)を最も一般的なif文の構造の例として，最初に教える．次に，(3)のなかで条件命題が偽の時に処理する文がないという特殊な例として，else以下のないものを教えるのである．もっとも，ここでいう一般と特殊の関係は，無条件に一義的に定まるものではない．図5の指導順序の根底には，条件命題が真の時のみに何らかの処理を行うifとthenだけの文を一般と考え，偽の時にも処理したい文がある場合を特殊ケースとするという論理がある．この論理

目次に戻る
前のページへ
次のページへ

変数

Visible を false にしたのですから、Height や Width, Color などのプロパティにはどんな値を入れても意味がないですね。よく考えると、Text プロパティの値さえしっかり残されていれば、他のプロパティはどうなっても、極端な話なくなっても構わないのです。中継用入力欄を使わない、もっとよい方法はないのでしょうか。

ここで、新しい方法を試してみます。中継用入力欄(EditTemp)を削除し、プログラムを次のように変えてください。

```
procedure TForm1.ButtonChangeClick(Sender: TObject);
var
  temp : String;
begin
  temp := Edit1.Text;
  Edit1.Text := Edit2.Text;
  Edit2.Text := temp;
end;
```

このプログラムを解説します。

最初に出てくる「temp : String;」は、中継用入力欄(EditTemp)の代わりに「temp」という入れ物を作ります、という宣言です （変数宣言の前には、「これから変数を宣言するよ」という意味で「var」と書きます）。宣言といわれてもピンとこないかもしれませんが、これはフォームに入力欄を置くのと同じことです。ただし、入力欄と違い、この「temp」という入れ物は、そのままでは画面に表示されません。このような入れ物を変数といいます。入力欄(やボタン)も実は変数の一種ですが、この「temp」という変数はより原始的な変数です。

もちろん、「入力欄の内容の入れ替え」の時のプログラムのように、このような原始的変数を用意せずに中継用入力欄を使ったままでもプログラムは正しく動きます。中継用入力欄の Visible プロパティを false にしてやれば、画面にも現れなくなります。しかし、画面に現れない中継用入力欄にとっては、Height や Width などのプロパティは意味がありません。結局、中継用入力欄にとって必要なものは、Text プロパティの値を保持することだけなのです。だから、「temp」という原始的な変数を用いることは、プログラムにとって必要充分な機能を提供することになります。こうして必要充分なものを用いるようにすると、(僅かですが) プログラムが速くなり、プログラムサイズが小さくなります。

図4 「変数」

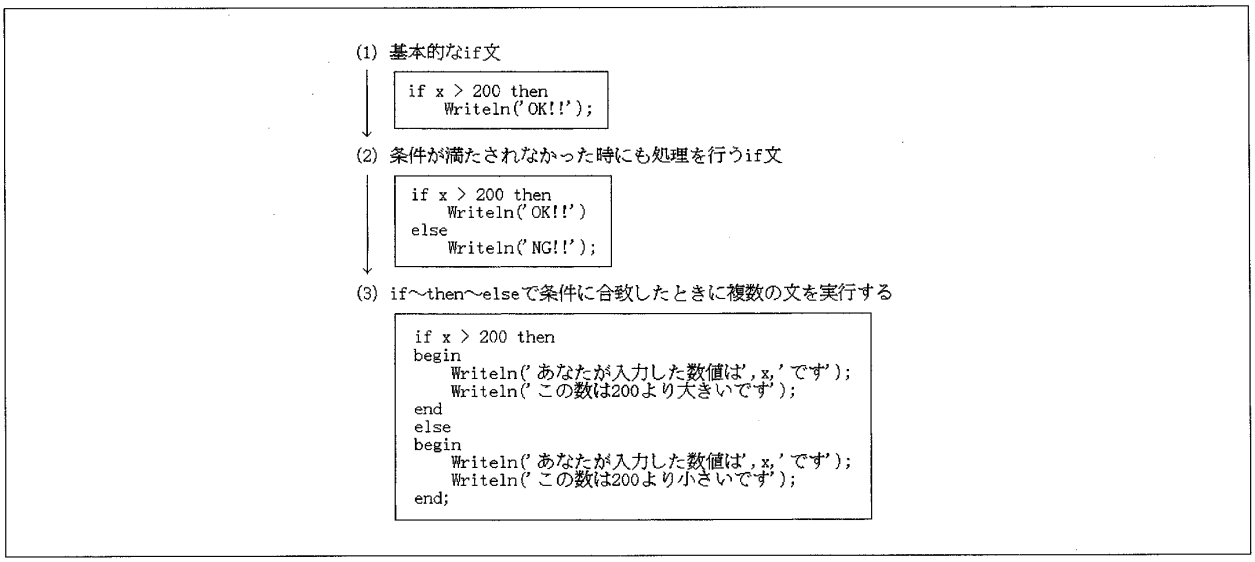


図5 藤本（1996）における if 文指導の流れ

は，必ずしも誤りとはいえない．しかし，我々は，if 文を〝完全に〟教えることを目標としているのである．その視点からすると，「if～then～else」すべてが揃ったタイプと，「if～then」だけのタイプとではどちらが一般でどちらが特殊かは確定する．すべてが揃ったタイプが一般であり，else 以下のないのはその退化した特殊な場合であると考え，教える順序もそれに従った方が，(目標達成に至るまでの）学習者の負担が少ないと，我々は考える．

ところで，then，else 以下で処理する文が１つか複数かという問題は，else 以下のある／なしの構造の問題と比べれば些末であるので，ここまで後回しにしてきた．１つの場合と複数の場合の関係については，複数の場合を一般と考える．この考えは第一に，数の本質から容易に支持されよう．第二の根拠は，複数の場合には処理する文を「begin～end」の形で囲まなければならないという〝手間〟の多さの問題である．手間のかかることならば，そしてそれでもどうしても教えなければならないのであれば，そちらの方を一般として最初から教える方が効果的である．

我々は，さらに一歩進んで，処理する文が１つしかない時にも begin と end を消さずに，残しておくように教えたいと考える．理由は，２つある．第一に，処理する文が１つになったとしても，文法的には begin と end を消す必要はない．さらに，実際のプログラミングの場面では，最初は実行文が一つでも，後に文を加え複合文になる事がよくあるが，その様な場合，begin と end の間に，新たな文を付加するだけで良い．対して，複合文の場合にのみ begin と end を新たにつける様に指導すると，多くの学生がこれにつまずき，結果として習熟に時間がかかるという傾向がみられる．それならば，処理文の単数・複数に関係なく，begin と end を残しておく方がよいのではないか．省略できるものは省略した方が，プログラムを少しでも短くするという観点からは好ましい，という考えもあるが，そこに殊更にこだわる必要はないと思われる．第二に，then と else がそろったタイプの場合に，例えば図５の(3)において，仮に then 以下の処理文が単数になったからといって，そこから単に begin と end を消すと，else の上の文に〝文の区切り〟を意味する「；(セミコロン)」が残るためにエラーが発生する．これは PASCAL 言語の文法に起因するエラーであり，ある意味では瑣末な事なのであるが，学生の指導上，頻繁に出くわすエラーである．

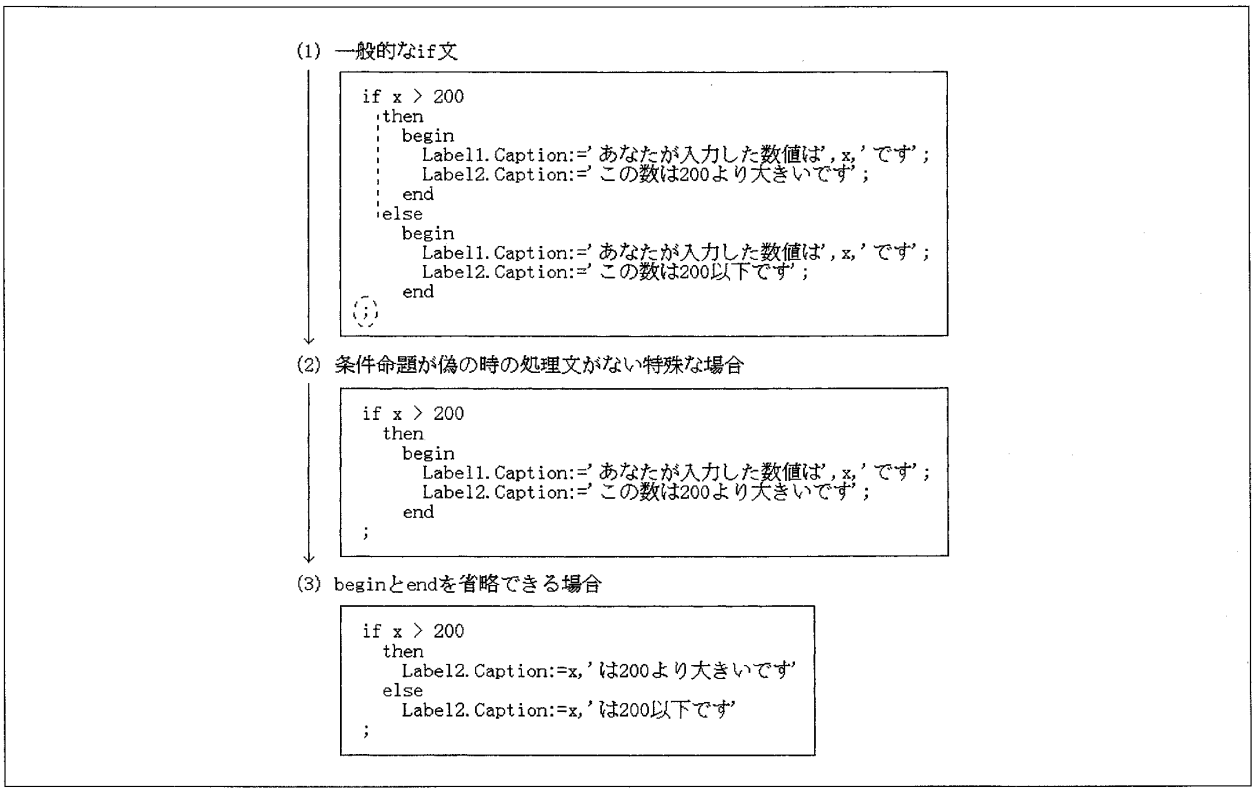


図6 if 文の指導順序

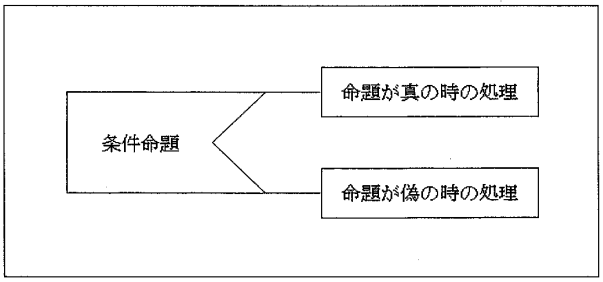


図7 PAD による条件分岐の記述

このように瑣末に思える事ではあるが，未然に〝余計な〟エラーを防ぐ事が出来，より本質的な点に学生の関心を向かわせる事ができるのならば，その様に指導する事は意味があるのではないだろうか．

もっとも，ほとんどのテキストでは，処理文が1つの時には begin と end のない形を掲載している．そこで，学生がそれらのテキストを見た時に混乱しないようにするという配慮は，必要であろう．したがって，「then, else それぞれの場合で処理する文が1つしかないときは，その前後の begin と end を省略することが可能である」と，簡単にふれる程度にしたい．

以上のことから，我々の構想する指導順序は，図6のようにまとめられる．図6を参照しながら，if 文の指導について書き残した2点について述べる．

1点目として，我々は，(1)に示したように，条件命題と then の間で改行し，then と else の記述位置を揃えて字下げするべきであると主張する．then と else の違いは，条件命題が真の時に実行されるか偽の時に実行されるかの違いでしかない．そして，命題における真と偽という関係は，本来全く対等なものである．したがって，図5の(3)のように，then を条件命題のすぐ右に書いて else との論理的な対等関係を見えにくくするのは妥当ではない．実際，プログラムの論理構造を的確に表現するために開発された PAD（Problem Analysis Diagram）では，条件分岐を図7のように書く．命題が真の時の処理と偽の時の処理は，対等な位置（論理レベル）に置かれている．図6の(1)のように then 節と else 節

を同じ位置にする，すなわち論理的対等関係を明瞭にする形に書くことによって，PADを用いることなく，PADのもつ論理構造の視覚的明晰性を獲得できるのである．結果として，「一般のif文ではthen節とelse節のうち必ず1つが実行されしかも1つしか実行されない」という，if文の本質的理解も容易になる．

2点目として，上の論理的構造の視覚化という観点に関連して，if文の終わりのセミコロンは，else節の終わりにあるendの右ではなく改行してからつける方がよいと思われる．endの右につけたままの状態から，else以下のないタイプに退化させるとすると，セミコロンごと消してしまう誤りが生じやすい．改行してからセミコロンをつけることによって，このセミコロンがelse節の終わりではなくあくまでも"if文の終わり"を告げるものであり，したがって欠かすことができないという，論理構造が分かりやすくなるのである．

4．今後の課題

第2章で提示された講義案は，半期用（半年用）のもので，本稿執筆段階では，まだ構想段階である．今後，この講義案の具体的な内容を検討した上で，次年度（2000年度）のプログラミングBにおいて実施する予定である．その上で，学生側の学習理解度を適宜チェックし，我々の案が，本当に学習側の負担を軽減し，無理なく学べるようになっているのかを検討する．なお，本講義案は，自学自習できる事を前提として作成しているので，学生が講義時間以外でも自由にアクセス出来るように，教材をWeb上にオンラインテキストとして用意する予定である．プログラミング教育におけるこのような形式の有効性についても，学生へのアンケート調査等を通じて調べる予定である．それらの結果については，講義案の総括として，次年度公表したい．

注

(1) ここに，ビジュアル・プログラミング環境とは，Windows上でのアプリケーションで用いられるボタンやメニュー等を予めライブラリとして用意し，開発者がそれらをフォームと呼ばれる基盤の上に貼り付けるという作業でアプリケーションを開発して行く，環境である．この様にインターフェース部分は処理系が用意しているので，開発者は，それらに対する動作が行われた（イベントが発生した）とき，例えば，ボタンクリックやメニュー選択などがなされた時の処理内容をコーディングするという作業に専念できる．

(2) Delphiでは，注(1)で述べた，ビジュアル・プログラミング環境を実現するために用意されたライブラリ（ボタンやメニュー等）をビジュアル・コンポーネント・ライブラリ（VCL）と呼ぶ．因みに，マイクロソフト社のVisual BASICやVisual C++では，対応するものをMFC（Microsoft Foundation Class）と呼ぶ．

(3) 従来型のプログラミングからビジュアル・プログラミングに移行した際に，学生が最も大きなギャップを感ずる点は，マウスをクリックするというイベント（動作）に対して何らかの処理を行う，という様な「イベント駆動型」プログラムの概念である．多くの学生にとって，「イベント発生情報のキャッチは当該コンポーネントが行い，開発者が行うのはそのイベントに対する処理内容のプログラミングである」というスキームに馴染むまで，時間を要する様である．

(4) 入力欄（TEdit）はオブジェクトであるが，コントロールと呼ばれることが多い．Delphiでは，様々なオブジェクトの中でも，コンポーネントパレットに載っていてフォーム上に簡単に配置できるものをコンポーネントという．コンポーネントの中でも，入力欄やボタンのようにプログラム実行時に外観が表示されるものをコントロールという．

(5) このような原則を立てることは，教育学の世

界では決して新しいことではない．例えば，数学教育においては，有理数の四則計算について，典型的な問題から出発して他の問題はその退化した形として系統的に教えようとする「水道方式」による指導過程がある．本稿で述べる講義案の作成に当たっては，「水道方式」からヒントを得るところが大きい．

(6) 市販のDelphiのテキストでは，変数とコントロールがうまく関連付けられていない．「はじめてのプログラム」などと称して入力欄やその他のコントロールを使ったプログラミングを最初に経験させるものの，その経験は全く生かされずに旧態依然とした変数の説明が始まっている．そのため，GUIという(学習者の関心をひきやすい)恵まれた環境を生かせずにコンソールプログラムで説明を行っているテキストも少なくない．

文献

森田 彦・新國三千代・原田 融 (1993)「アルゴリズム理解能力の分析」『社会情報』Vol.2, No.2：87-99

森田 彦 (1995)「アルゴリズム理解能力の分析II」『社会情報』Vol.4, No.2：95-104

三原幸一(1999)『基礎からわかるDelphi 5』秀和システム

藤本 壱(1996)『Delphiで作るWindowsプログラム』サイエンス社