入門プログラミング教育につづく科目案

都倉信樹†

概要： 大学の最初のプログラミング入門につづくプログラミング科目は種々のものが提供されている．それらに１つの案を追加する．内容は新言語＋開発環境への移行促進のための多数の小さいプログラム作成（ブリッジ）と第三者が使用することを前提としたプログラムプロジェクトを，順に工夫を重ねつつ開発することを狙っている．後者の課題にはいろいろの候補が考えられるが，ここでは著者が必要とした「木を描画するプログラム」の開発を例としてあげて，この種の科目の内容案を紹介する．

キーワード： プログラミング入門につづく教育，木描画プログラム，単純な木の表現法

A Programming Course Plan as a Post-Introductory Programming Course

Nobuki TOKURA

Abstract: A proposal of an advanced programming courses with two main elements. The first element is a bridge between elementary C programming course and advanced course in which several desktop applicaitons are realized, using C# and Visual Studio in proper way. Students acquire firm programming skills and communication skills. In the second project element, challenging desktop applications are targeted. Students are expected to experience the joy of programming. As an example, a TreeEditor project using simple level-data representation is described.

Keywors:: Post-Introductory Programming Course, Tree drawing program, a simple tree representation

１．はじめに

プログラミング教育は今後大きな変革を経験すると予想される．義務教育段階や大学の入門的科目についての議論は情報処理学会，また，当研究会で盛んに議論されている．小中高でのプログラミング教育の開始・強化に伴い，大学でのプログラミング教育も影響なしとはしない．ここでは，大学での入門的なプログラミング教育につづいて行うプログラミング教育についての一案の報告である．実践報告ではなく，こういう教育にも目を向けようという提案である．

「プログラミング入門につづくプログラミング教育」をどうするかということである．「」の長たらしい用語は避けたいので，乱暴だが，入門プログラミング教育をプロ１，その後に続くプログラミング教育をプロ２と呼ぶことにする．これは雑な用語で，プロ２は実際はいろいろの類型の講義・演習が行われている．大学でプログラミング力を付けたいという意欲を持って入学した学生は，入門科目は易しすぎ，かといって，プロ２になると，関数型言語などの高尚な講義になってしまったりして，意欲を失ったりする．入門科目で多くの学生がプログラミングを嫌になるということも，若い先生方から聞いて驚愕したが，折角情報関係学科に入学したのに，プログラミングが嫌になる学生を多数生み出すのは残念である．本文はその対策とは少し違う方向の議論である．少数かも知れないが，プログラミングに意欲をもって入学した学生にさらに興味を持って取り組んでもらって，プログラミング力をしっかり付けていただくことを狙った教育を検討したい．

この科目の内容は大きくみると２つの要素を用意する．第１の要素（ブリッジ）は，プロ１から新しい開発環境，言語とやや組織的なプログラミング活動を行う習慣つけ，あるいはマインドセットの形成も意図する．第２の要素（プロジェクト）は，実際のユーザの存在を前提としたアプリケーションの開発プロジェクトである．

前提として，プロ１で基本的なプログラミングは経験しているとする．この科目では Visual Studio(VS と略記)で，C# 言語を使い，Windows フォームアプリケーションを作成するとする．

２．ブリッジ

プロ１で C で基本的なプログラミングを経験した学生が興味を持って取り組んでくれ，さらに大きなプロジェクト課題を実際に実現できる力を短期間に集中的に備えさせ

† 大阪電気通信大学
Osaka Electro-Communication University

るためにどうすればよいかが,著者の設定した問題である．これを行う要素をブリッジ(橋渡し）と呼んでいる．それは VS と C#の集中講義をすればよいか？おそらく NO であろう．しかし，いろいろの試行錯誤を経て，著者が手応えを感じた方法をここに紹介したい．

2.1 概略

学生はプロ１で,C を使った基本的な演習を行ってきて,未経験な C#と，VS という統合環境を使うとする．最近は玉石混淆だが，C#の解説書もインターネットにも大量の情報がある．基本的には，仕様を提示してプログラムを作らせるというのでなく，調査し，あるいは，テストしてみるという形で，自力で必要な情報を獲得し，それを利用してプログラムを実現する力を付ける．

ただ，これを漫然とやってはだめで，つぎのようなことを学生に求める．

［調査］ 必要なことは徹底的に調べる．C#なんとかで検索すると多数のページが見つかる．まさに玉石混淆であることは気づくだろうが，ここから真に役立つ情報を見つけるよう心がけ，出所とともに要点を記録する．Microsoft の DOCS が基本的であるが，機械翻訳の部分は全く意味不明なことも多い．このとき，英語に切り替えるとちゃんと分かることを学生にも経験して貰いたい．英単語が分からなくてもそれを教えてくれるサイトもある．それも自分のドキュメントに控えておくとよい．役立ちそうなサンプルコードが見つかることもあるが，これがまたいろいろ面白い．よく吟味してテストしてみる必要もある．

すでにいい方法が知られているなら，それを使わせて貰えばいいので，すべてにオリジナリティを求める必要はない．安直な課題を出すとコピペで済むこともあるが，なかなかインターネットを探しても直接使えるコードは見当たらないのが実際である．ここで頭をひねらないといけないことになる．説明がまずくていまひとつ分からないコードも多々ある．その解読のために，さらに調査を続けることもどんどんやればいい．今回試行した課題の多くはそのまま使えるサンプルはなかった．

自分のやりたいことを簡単な検索ワードで表現することも難しいが，表現出来てもそこで出てくる情報でぴたりということはほとんどない．しかし，求めれば見つかるようになるし，いつもそれが頭にあると，不思議と欲しい情報にさっとたどり着けたり，散歩や入浴中に自分でアイデアを見つけることもある．経験者ならすぐ答えてくれるようなごく簡単なことでも未経験な世界で探すとなるとそう容易ではない．しかし，いつもこれをどうするのかと目の前にぶら下げていると，若干時間はかかっても解決が手招きしてくれる．こうして解決できたことは，非常に喜びを与えてくれ，プログラミングの面白さの1つになる．

[コード] プログラムをエディットする段階で VS の便利な機能を活用するように誘導する．種々のヒントで新しいことを学ぶことも多い．

[テスト] テストは熟練を要することではあるが，最初つくるプログラムではそう複雑なことは要らない．その段階からテスト，デバッグのために用意された機能になれておくことが望ましい．また，実験もしばしば行う．

[コードの説明] 教員としばしば面接してコードレビューを受けることになる．そのために，コードとその説明，UI 設計書，実行画面などを用意する．説明はどう書けばよいかなかなか難しい．あとでも簡単なコードとその説明，UI 設計書の例を示しているが，なかなかこれでよいという線がわかりにくいのは実際である．1つの判定基準は，教員や第三者に説明するときに，その説明が機能したかどうかということがある．漏れがあったり，逆に過剰な説明もよくない．自分の書いた説明書の不備を自分で発見することで，次はどういう書き方をするかを学生自身が自得することを期待する．どうしてもそういう点が弱い学生には，指導者がある程度の改善指針を示すことは有用である．

[プラン] 解決したいという目標や構想を表すことを表す用語としてプランとここでは呼ぶ．たとえば，ラベルの表示を実行時に変えるにはどうするかなどの疑問はそれを解決したいという小さいプランになる．プログラミング過程で，大きな要求を解決するために，小さいプランがどんどん発生する．それを簡単な言葉でリストにどんどん登録して，次何をやるかを意識するようにする．この作業をいつもやってもらう．プランの粒度はさまざまである．それでいい．ひとつ解決したら，それから多数のプランが派生することもある．これをミーティングの際に共有して，つぎどれを検討するかなどの作戦を立てるのにも役立てる．

[報告, プレゼン] 指導者は学生が各自のプランをもとにどう作業が進行しているかの報告を聞いたり，コードの説明のプレゼンを聞いたりして，適切な指導をする．学生は他者の意見や説明を傾聴し，自分の意見も明確に述べて，よい解決に持って行く議論の仕方を意識して学んで貰いたい．以上，注文が多いが，こういう説明をタイミングをみて学生に浸透させていく．こういう汎用スキルは卒業研究で初めて学ぶのでなく，できるだけ早くから身につけてほしいと考えるからである．

2.2 具体的事例

この科目ではデスクトップアプリの開発をするが，もっとも基本から出発するとして，以下プランの並びとして紹介してみたい．

ここでは，Form だけ，あるいは，Form の上にラベルというコントロール1つ，のちにラベルの数を増やすこともある．C#では多数の便利なコントロールが用意されていて，これらのコントロールを上手く使って，GUI を実現するが，敢えて，ラベルというコントロールしか使ってはい

けない（すくなくとも初めのうちは...）という厳しい制約のもとで一体どういう機能を果たせるかを考えて，プランを作っていくというやや不思議な設定でブリッジを構成した．

Label のもっとも基本的で重要な役割は，画面上に文字列を表示することである．これから Label の Text の機能をできるだけ引き出してみよう．実際は学生とインタラクティブにやっていくが，ここでは順調に作業が進んだとして，その経過を記してみる．

［Label_Text　プロジェクトを作る］

P1.　Visual Studio を起動して，Solution　“Label_Text” を新規作成する．→ Form1.cs[デザイン]　とタブのついたウィンドウ(以下，単にデザイン画面）が開く．ここに，初期フォーム画面がある．

P2.　Form1 のプロパティウィンドウが開いているので，２つのプロパティを次のように設定する．

Font の右の方をクリックして，FontDialog でメイリオ，サイズ 11 を選択し，OK とすると，画面が少し大きくなる．Font のサイズはきっちり 11 でなく，11.25pt になっている．システムが適当に設定しているので，これはお任せする．

Form1 の Text プロパティを，このプロジェクトの名前，「Label_Text」に変更する．

P3.　(Test)　開始ボタンをクリックする．

→　図 1 が現れる．そこで，問いかける．

Q(P3)　このプログラムでは何が出来るだろう？

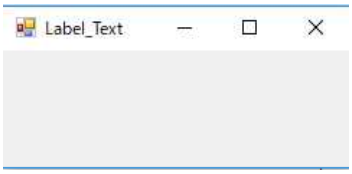


図 1　実行画面　Figure1 Test run

A(P3) 最小化，最大化，閉じる，サイズと位置の変更．

こういう基本的な概念などもテストの際に理解させる．

P4.　Label の追加．ここで，デザイン画面の左のツールボックスから Label をみつけ，これを Form1 の上に持って行く．→ label1 と表示された箱が現れる．

ここで，プロパティウィンドウのタイトルに，label1 System.Windows.Forms.Label が表示されるようにする．これは複数の方法があり，見つけた方法を報告して貰う．そのプロパティウィンドウをよく観察する．沢山のプロパティが並んでいるが，ここで注目するのは，　Font,Text　の２つ．Font は Form1 に設定したのと同じになっている．label1 は Form1 の上に持ってきたので，Form1 の子供であり，親と同じ Font が自動的に設定されている（Q.Why?　）

label1 はシステムが自動的に付けてくれた名前だが，これは(Name)のところを書き換えれば改名できる．そこで，試しに短く L1 としてみよう．なにか変化はあるだろうか．プロパティウィンドウの見出しが，L1 なんとかに変わっている．これでプログラム中，このラベルの名前は label1 でなく L1 に変ったことが確認できる．ところがデザインの方の箱は label1 のままである．L1 のプロパティをみると，Text のところが label1 のままだからである．

P5.　Text を"Hello,World!"に変えるとどうなるだろうか．デザイン画面のラベルは"Hello,World!"に変わる．

P6(Test)　開始ボタンで実行する．何が起こるか．次の図 2 のようなウィンドゥが現れる．

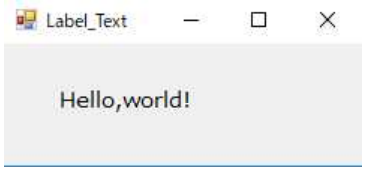


図 2　P6 テスト画面　Figure 2 P6 Test run

P7.　Label は画面に文字列を表示するための基本的なものである．しかし，いつもこのワンパタンの表示では面白くない．

「表示をどういう風に変えたいかを各人 10 通りを目標に提案してください．メモ帳に書き込んでおくように．」

P8　時間があれば，どれを作るか議論する．次回までに２つのプログラムを作ってくる．

P9　なかなかラベル 1 つの道具立てでは出来ることが限られていて，行き詰まったところで，表示が変わるような課題として，現在時刻を表示するというプランがあったのを取り上げ，それをどう実現するか議論する．

P10　起動時，Form1()コンストラクタの中で，L1.Text に時刻を表示する方法をやってみる．この時刻を表示する方法をプランとして，調査したり検討する．

うまく動くところまで行ったが，起動ごとに時刻を表示するがあとは変わらないので面白くない．せめてディジタル時計のように刻々変わる表示はしたいというプランが出る．

P10.　これをどう実現するかを調べてきて，Timer コンポーネントを使うという方法を知る．そして，tick イベントハンドラの中で，現在時刻を表示するという方針で，実現法を模索した．図 3 はその 1 つの実行例であり，毎秒時刻が更新される．

ここに掲載するためもっともコンパクトなコードを選んで次に示す．プログラムを作ると必ず説明を付けることを求めており，下にその説明がある．

毎秒更新される

図 3 テスト　Figure 3 Test run

ソースコード，その説明，実行画面に加えて，プログラムがどういうコントロールを使い，そのプロパティをどう設定している

かとイベントに対し結び付けられるハンドラの名前を表にまとめたもの(UI 設計書)も作成提出を求める．図 6.

Label_Text.sln　Form1.cs　(in C#)

```
1  using System;
2  using System.Windows.Forms;
3  namespace Label_Text
4  {
5      public partial class Form1 : Form
6      {
7          public Form1() => InitializeComponent();
8          private void timer1_Tick(object sender, EventArgs e)
             => L1.Text = $"{DateTime.Now}";
9      }
10 }
```

Fig 4 コード例
Figure 4 Code example

説明

1,2	using ディレクティブ　2 つの名前空間を使用する
3	このプログラムの名前空間
5-10	Form1 クラス　L.7,8 の=>はラムダ演算子．右辺の式を実行する
7	Form1 のコンストラクタ．InitializeComponent()は L1,timer,Form1 のプロパティの設定等を行う．そのメソッドは，Form1.Designers.cs という自動生成された別ファイルに記述されている．
8	tick は 100ms ごとに起こるように設定している．DateTime.Now で得た時刻情報をラベルに表示する．$ ”” は文字列補間式.

図 5　コードの説明例
Figure 5 Comments

No	On	Name	Control	Property/Event	Value
1		Form1	Form	Font	メイリオ, 11.25pt
				Text	Label Text
2	1	timer1	Timer	Enabled	true
				Interval	100
				Tick event	timer1_Tick
3	1	L1	label	(Name)	L1
				Font	メイリオ, 11.25pt
				Text	Hello, world!

図 6　UI 設計書　Figure 6　Design of UI

使用するすべてのコントロール類に一連番号を付ける．3 番のラベルは 1 の上に置くので，On　には 1 を記入する．L1 は Form1 の子と扱われる（このラベルをドラグしても Form1 の領域外にでることができないことを実験．）

1 つプログラムができると，いろいろの条件を変えてどうなるかを試してみること（実験）が有用である．

[実験]　テスト案を出し合って，実際に試して効果を調べる．

1　Autosize の効果を確認する．
2　Font 名だけでなく，スタイルやサイズを変えてみる．
3　ForeColor，Backcolor を変更してみる．
4　境界線を見せる．
5　DateTime の機能を調査し，実行例以外の表示も行う．
6　つねに自分でプランを増やしていくように勧めるが，時にはつぎのステップにつながるように，こんなのが欲しいというものを示して，プランづくりを促進することもある．たとえば，図 f7 はそういう意図で作って見た．

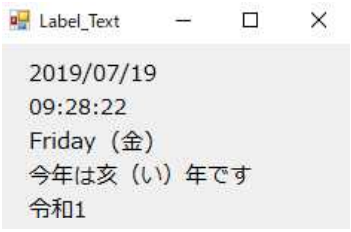


図 7 こんな表示して欲しい　Figure 7 I want...

時刻関係で十分いろいろのプランがでるが，これ以外に各種のシステム情報の表示などのプランも面白い．

ここまで，Label.Text について多数のプランを提案し，可能なものから実現していく過程を提示した．難易度はほぼ適正で，C#に関わる基本的な知識をこれに絡めて伝えることができる．

これに続いて，ラベルの移動を取り上げるが，これでマウスが登場する．ラベルのドラグ&ドロップはいい例題で，これでマウスの扱いはほぼ習得できる．ラベルに image を貼り付けて，円が動くようにする．また，ランダムに動かすことを考え，衝突検知の問題を扱ったりする．ゲームの基本と感じるからか，この辺りになると学生から多数のプランが提案される．これらをいつも目の前にぶら下げておくと，どんどんコーディングしていくようになる．

また，ラベルで他のコントロールの振りをするというプランが出始め，これはラベルと他のコントロールの機能の差異に目が向くきっかけとなる．単純なラベルだけに絞ったことで，そのプロパティやイベントを嫌でも丁寧に見るようになった．基本機能をじっくり調べ，自分なりの方法を見つける例が増えてくる．これはプログラミングの楽しさを味うことにつながる．この路線はやっていて楽しいので他にもいろいろの展開を考えたい．

３．プロジェクト

2のブリッジを渡れた頃（あるいは，並行してくさび型で)，プロジェクトを開始する．これはあるアプリケーションを個人あるいは小グループでPBL方式で開発する．ここでは，どういう課題を設定するかが悩ましいところで，ここでは，課題（大プラン）の１例を示す．

それは木を描画するプログラム(TreeEditorと呼ぶ)の開発である．おそらく探せばこの種のツールは多数あるだろうから,いまさら作るまでもないと考えるかもしれないが，そこは

(1)　演習課題である（新規性を狙うものではない．狙えたらいいが.）

(2)　類似プログラムが存在することは参考にできるものがあるので悪いことではない．

(3)　なにより，課題の提出者(この場合筆者)が，何かいいプログラムはないかと求めているものである．ということは，ユーザが確実に一人は見込める．

など，じわじわ説明して，やろうという気になっていただく．

(3)は実情を説明すると，納得してくれるだろう．求めることは，図8のようなラフスケッチを描いて，それをデータとして簡単に入力し，出来れば図9のようにちょっとましな図にしてくれればよいと説明する．図9ならドロー系のソフトで30分程度で描けるというかも知れない．ある教材を作るときに，10数以上の木を描く必要があったが，たまにしか使わないドローソフトであり使い方を忘れていて苦しんだ．そのとき，木のデータを入力したら，標準的な方法で図を描いてくれるプログラムが欲しいと思ったのが，ここにつながる．これくらいの発注者の希望を述べるだけであり，これも仕様ではなく大まかなプランである．

説明後学生は課題をじっくり考えて，なにをすべきかの構想(プラン)案をつくり，それをもって，教員と共同ミーティングをする．ここで，どこから手を付けるべきかを議論して，より詳細なプランにブレークダウンしていく．

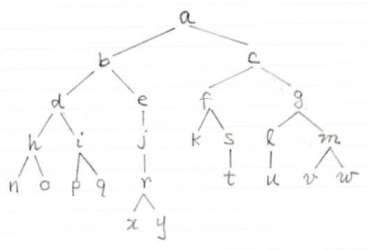


図8　　鉛筆と定規で描いたラフスケッチ

Figure 8　A rough sketch of tree (pencil and rule)

なお,学生(群)ごとに異なる課題を設定する場合,この辺りで教員は忙しくなるので，2 から 3 への移行は少し時間をずらして行うことになろう．

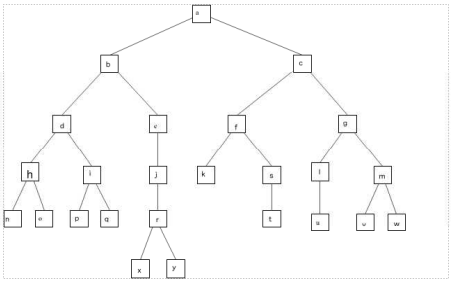


図9　　描画結果　　Figure 9　Output

４．　木を描画するプログラムを作りたい

図8のようなスケッチからグラフィカルな木表現を求めるには，木を記述する情報を入力してやらねばならない．グラフ理論的な問題をプログラム的に扱うには種々のデータ表現法がある，もっとも基本的なものを考えると，枝のリスト,各ノードの子のリストを与えるなどが考えられる．

4.1　木をどうやって入力するか

[A]　枝リスト

枝リストでは，枝をノード名の順序対として表す．たとえば，(a,b)順序対は，ノードaからノードbへの枝を表す．ただ，すべての枝を集めただけだと，根の指定とか，子の順序などの情報は得られない．そこで，次のような約束で枝のリストを作る．

(1)　リストの先頭に書かれた対の第一要素が根である．

(2)　同じノードｘから複数の枝がでるときは，(x,y),(x,z)のようにxで始まる対を並べる．その際，子ノードの順序が対の要素の第二要素の順で分かるようにする．

```
{(a,b),(a,c),(b,d),(b,e),(c,f),(c,g),(d,h),(d,i),(e,j),(f,k),(f,s),(g,l),
(g,m),(h,n),(h,o),(i,p),(i,q),(j,r),(s,t),(l,u),(m,v),(m,w),(r,x),(r,y)}
```

図 10　木表現　ノード対のリスト

Figure 10　Tree representation using a list of node pairs

[B]　親と子ノードリスト

これもリストの先頭に書かれた対の第一要素が根となる．親のノードの後に，子供のノードを列挙する．これは子の数が事前に固定されないような場合でも対応出来る柔軟性のあるデータ表現である．

```
{ (a : b, c), (b : d, e), (c : f, g), (d : h, i),
  (e : j), (f : k, s), (g : l, m), (h : n, o), (i : p, q),
  (j : r); (l : u), (m : v, w), (r : x, y), (s : t) }
```

図 11 木表現　親のノードに対して，子ノードのリストを並べたもののリスト

Figure 11　Tree represetation using a list of node with a list of childrens

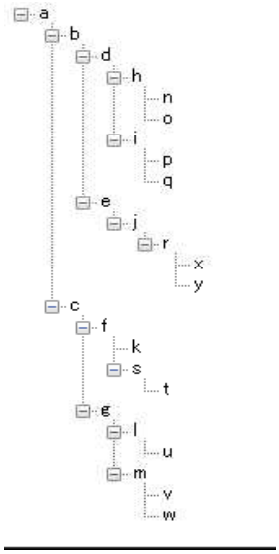


図 12 ツリービュー
Figure 12 TreeView

```
<?xml version="1.0"
encoding="us-ascii"?><TreeView><node
text="a"><node text="b"><node
text="d"><node text="h"><node
text="n"/><node text="o"/></node><node
text="i" ><node text="p"/><node
text="q"/></node></node><node
text="e"><node text="j"><node text="r"
><node text="x"/><node
text="y"/></node></node></node></node>
<node text="c" ><node text="f"><node
text="k"/><node text="s" ><node
text="t"/></node></node><node
text="g"><node text="l"><node
text="u"/></node><node text="m"><node
text="v"/><nodetext="w"/></node></node>
</node></node></TreeView>
```

図 13 XML
Figure 13 XML represetation

```
a
    b
        d
            h
                n
                o
            i
                p
                q
        e
            j
                r
                    x
                    y
        c
            f
                k
                    s
                        t
        g
            l
                u
            m
                v
                w
```

図 14 インデント方式
Figure 14 Indentationrepresentation
→

図 14

```
0,a
1,b
2,d
3,h
4,n
4,o
3,i
4,p
4,q
2,e
3,j
4,r
5,x
5,y
1,c
2,f
3,k
3,s
4,t
2,g
3,l
4,u
3,m
4,v
4,w
```

図 15 レベル-データ表現
Figure 15 Level-Data representation
→ →

図 15

[C] 木の階層を視覚的に表現する方法(TreeView 方式)

他に木の表現として，ファイルシステムのディレクトリ（フォルダ）の階層構造を表すのに，図 12 のような図的な表現がある．この表現に使う TreeView コントロールを利用して，簡単なエディタは作ることはできる．それで順に節点を追加して一から木を書きあげることも出来る．そうして出来た木の情報から図 9 のようなグラフ表現を書くことはできる．したがって，これを木の情報の入力用に使うことは可能である．ただ,これも入力はかなり面倒であり，複雑な木構造を長時間掛けて編集してもそれはプログラムを止めると消えてしまう．ここにシリアル化(serialization)という手段がある．XML 等のテキストファイルとして保存することで，プログラムの寿命を超え，永続化が達成される．この TreeView のデータをシリアル化した結果図 13 をみてみよう．べたづめでなく，階層構造が把握しやすい形に表現することも出来るが，スペースの都合で省略する．(XML の 1 つの難点は字数が多く，スペースを食うことである)．

これは木構造を表現するために，完全括弧付きの構文木を使うのと似ていて，たとえば，節点は<node> </node>という括弧で括っている．なお，<node text="w" />という簡略形も使われている．最初からこれを書けと言われると非常に困る．

もっと簡単な方法はないかと考えた．そのとき，閉括弧的なものを使わずインデントで階層構造を表す方法が Python などで使われているのに着目した．

図 14 の形で入力して木構造を描画するプログラムを作成してみた．やはり入力はこれくらい簡単だとテストデータも沢山作れるし，実際に多くの木を描く必要がある状況ではこの方法は魅力的と感じた．

しかし，図 14 の表現をみてすこし危惧を抱く人もあるかもしれない．tab をすこし間違えると木構造が崩れてしまう可能性がある．それと複雑な木だと深さがどんどん深くなって，画面でもプリントアウトでも折り返しをどうするかが問題となる．

折り返しが入るととたんにわからなくなるし，ページをわたると戸惑うこともある．タブは white space で見えないからである．それを可視化する方法をいくつか検討し，タブの数（木のレベル）を行頭に書き，カンマなど適当な区切り記号の後に，その節点に関わる情報を書くという方法を思い付いた．これを，**レベルーデータ表現**と呼ぶ．具体的には図15の表現になる．

図14, 15を見比べると，Tab区切りのインデント方式と全く同じ情報を含むことがわかる．そして，木のラフスケッチを見ながら，容易にテキストとして入力していくことが出来る．この利点は大きい．こういう素朴な発想である．

具体的な必要からこういう発想につながったわけで，実際の問題はやはり重要である．仕様でなく，要求から入ることで，学生は新しいアイデアを出しやすくなる．

図16は試作木エディタの実行画面例である．左端がテキストボックスで，ここに，テキストファイルから木表現を取り込み表示している．一行目はコメントで木の描画では読み飛ばす．

中のパネルは TreeView 表示で，右端には木をグラフとして描画している．

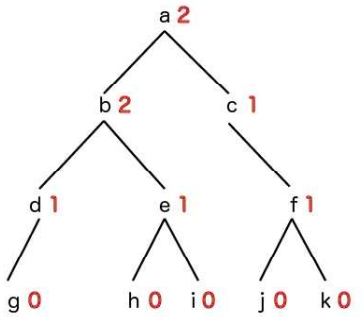
5． 拡張

多くの応用では，単に木を描画するだけでなく，ノードや枝のそばに付加情報を付けたい．たとえば，図17はアルファベットで表現されたノードのそばに，あるアルゴリズムで付加された情報が数字で表示されている．こういう木を表現するには，先のレベルーデータ表現のデータ部に，

一般的には

レベル：ノード情報(名前，属性情報など)

を1行に書いて，1つのノードを表す．



(a)属性情報の表示

```
0:a,2
1:b,2
2:d,1
3:g,0
2:e,1
3:h,0
3:i,0
1:c,1
2:f,1
3:j,0
3:k,0
```

(b)木表現

図 17　(a)属性情報のついた木, (b) レベルーデータ表現

Figure 17　(a)Tree with attribute data, (b)its level-data repuresentation

6． 次のステップ

当初望んでいた機能は７までできた．学期の時間との見合いでどこまで作るかは決まる．初めはなかなか順調には行かない．作りつつつねにこの先どういう機能が必要か改善点などをプランリストに書き込むことを勧める．そうすると，次の課題を明確化できる．

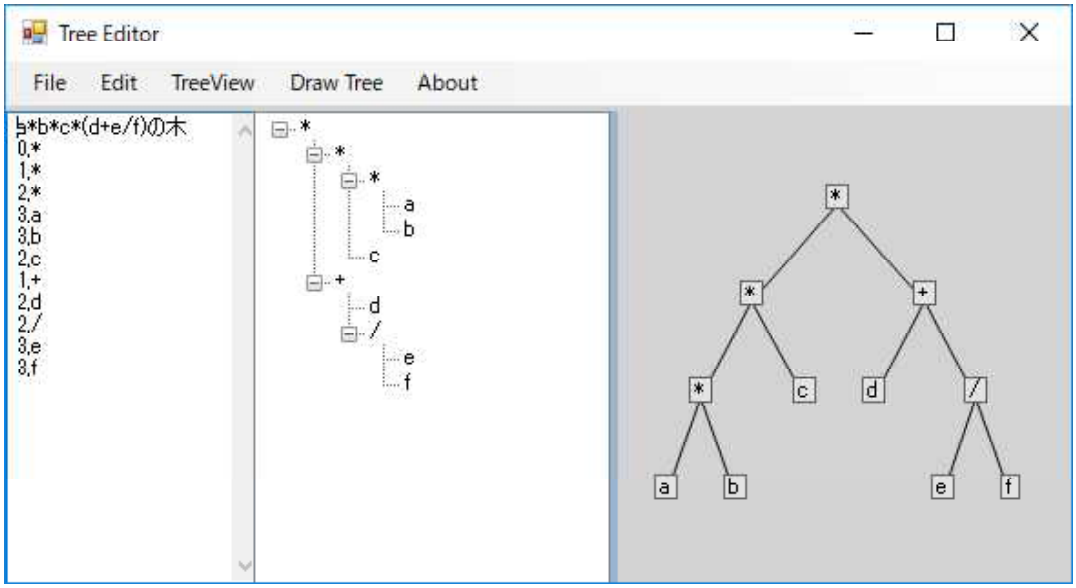


図 16　試作 TreeEditor の表示画面例

Figure 16　Screen shot of a level-data tree representation, a TreeView and a Graphical tree

このプロジェクトで残る検討事項の一部を紹介する．

(1) 節点や枝の色や線の幅，文字のフォントその他，木の表現をユーザが自由に設定できるようにする．その設定を保存する機能ももつ．

(2) 描画した木を印刷したり，ファイルに保存する機能

(3) このレベル-データ表現は非常にコンパクトに木を表現できるが，これで XML,JSON などのシリアル化の手法の代替ができる．XML よりコンパクトであり，変換や木の扱いもかなり簡単になる[1]. これは(1)の設定情報の保存と初期設定に使える．

(4) ここでは種々の機能を1つのプログラムに詰め込んだが，逆に有用な機能を取りだし他でも使えるようにプログラムを再構成する．いわゆる，マイクロサービス的な利用である．それを組み込んだ他のアプリケーションを作る．

７．プロ２について

プロ２について簡単に述べる．プロ２として，いくつかの学科の取り組みをみると，次のような類型が見られる．ひとつは，プロ１では C のごく一部と簡単なプログラミング演習しかできないので，プロ２は引きつづき C でポインタ，再帰などを活用して，データ構造や基本的アルゴリズムを教える．他の類型としては，非手続き型の言語，Scheme,SML,Haskel などに触れさせるものがある．これは情報専門学科なら当然含まれるべきものであろう．これも入門レベルに留まる．また，Python, Ruby, JavaScript などを使って新味を出そうとしているのもある．なお，プロ１でこれらの言語を使って入門し，プロ２で，Java につなぐという逆の方針もある．

ここで提案したのは，これらのプロ２とは多少異なる内容をもつと言えるであろう．

C#＋ VS でデスクトップアプリを作成する．異なる開発環境，言語，対象へできるだけ早期に移行し，かつ自力で以後のプログラム開発を進められるように種々の手立てを講じる．1つは自発性を引き出すために自分でプランを多数案出することを重視する．ブリッジでは，本稿の例ではラベルを徹底的に使いこなすという設定にしたのがプランを編み出すのにプラスの方向に働くという感触をもった．そして,プランを実現するために自力で情報を取捨選択し，そこから役立つものを見つけられるようにする．また，簡単なプランを多数やるが，簡単であるからコードレビューもしやすいが，ここで学生はきちんと自分のコードについて毎回説明を求められる．説明を円滑にするのにプログラムの説明をコード作成時に書く習慣を付ける等，コミュニケーション力も付けるようにサポートする．おそらくブリッジと続くプロジェクトはくさび型に実施するのがいいかと考える．プロジェクトはやはりかなり重くて，行き詰まりもあるだろう．ブリッジでやったような小さいが必ず何らかの新しい知見に結びつくものはやって達成感を得やすく，かつ後々使えるし，気分転換にもなる．

プロジェクトは教員が作って欲しいと思う課題を使うのがよいと考えている．アジャイル開発的な開発スタイルに自然になる．ここでも定期的に説明付のコードレビューを行う．ここでもつねに開発上発生した課題をプランとしてリストアップして教員もそれについて議論して，プロジェクトの完成に導く．このプロジェクトは一人または複数人のグループでPBL的に実施する．教員以外の人も含めた発表会を行い，プレゼン能力を高めていく．これを実施するには週2回で半期以上が適当かと考える．ただ，この種の科目は無理だという反論は予想される．ひとつは過密カリキュラムのためこんな時間を食う科目は入れられないという反論があろう．これは学生の能力の何を延ばすのか，学科の教育理念の問題である．また，ここはサイズ，時間的にもほどほどの例題を提示しているが，そんな適切な例題がないという反論もあろう．たとえば，ロボットを扱う学科なら常に新しい課題が生まれる．しかし，情報関連学科は意外と紺屋の白袴で,プログラムする題材がないという．これはそう心配しない．自分の講義で，なにかを分かりやすく教えたいとシミュレータを作ることは例になる．たとえば，著者はλ式の簡約化のシミュレータを作ってみた．当初は式変形でみせ，このレベルーデータ木表現を使うようになって，λ式を木で表して，木の変形としても見せるようにした．これも多数の例題をいちいち手計算で作っていたのに比べると,はるかに分かりやすく簡約過程が示せ,教材作成にも有効に使えている．このように題材の不足を嘆くことはないと考えている．

８．むすび

プログラミングを楽しむためにどういう方法がよいか，これは著者の1つの問題設定である．簡単に答えが出ないのは承知だが，学生と教員の二役でプラン誘導の方法を試行して，つぎつぎと新しいプランが発生し，プログラミングを楽しめるという感触を得ている．

教員から課題を与えるのでなくプランを共有して，ともにプログラミングを楽しむ教員の姿を見せるのもいいかと思う．先ずは教員がプログラミングを楽しむことが早いのかも知れない．

謝辞 査読者に：適切な査読コメントをいただいたことを感謝します．

参考文献

[1] 都倉信樹：ある単純な木表現法の応用について，FIT2019, 6D-7.(Sept. 2019).