抽象構文木プログラムの編集，実行，代書

都　倉　信　樹
（大阪電気通信大学）

概要： CE160 で「木構造プログラミングシステム」の概略を報告した[7]．　このシステムは

1．プログラムを木構造として，入力・編集・保存する．
2．具体構文から離れた抽象構文での思考を促進する．
3．木プログラムの編集系，その解釈実行系，他の言語に変換する代書系を備える．

本報告では，システムの説明と代書について説明する．
アルゴリズムから具体的プログラムへの橋渡しの教育での利用の可能性を検討する．

キーワード：　木構造プログラミング，　代書系，　アルゴリズム教育

１．はじめに

プログラミング教育に関して，多くのシステムが作られ，教育実践も本格化しつつある．本研究は中高大でのアルゴリズム教育の支援を目指している．その支援系としての木構造プログラミングシステムを試作し，[5][7]で概略を説明した．ここでは，そこで提唱した代書系についてより丁寧な説明を試みたい．

従来のアルゴリズム教育で用いられたツールには種々ある．当然，プログラム言語が最も重要なツールであるが，１つの難点は言語の習得も同時に求められることになり，学習者の負担がかなり大きく，指導者も苦労する．アルゴリズムそのものは言語に関わりないものであり，言語に依存しないアルゴリズムの説明も試みられる．たとえば，フローチャート，NSチャートなどの図式で，アイデアを示せるが，かならずしもうまく行かない．実際に動かさないとアルゴリズムの理解が深まらない．やはり，プログラム化して，実行することが求められる．しかし，実用言語はアルゴリズムの習得に必要なレベルを超えて多くの機能を有し，どうしても敷居が高く，かつ表記も細心の注意をしないとコンパイラから叱られる．初学者にはなかなか厳しいものとなる．こういう困難を解消しようと多くの努力がなされてきた．たとえば，高等学校情報科教授用の資料[6]では，Python，Javascript, Scratch,どんくり（DNCL)が扱われている．いずれもあまりプログラム言語学習の負担の掛からない方法で，プログラミングを学習することを目指している．他にも種々の提案や実践があるが，それを展望することは本報告の目的ではない．

ここでは，従来型のプログラミング環境とすこし趣の異なる木構造プログラミングシステムの説明から始める．

２．木構造とその表現

木構造は親ノードに子ノードがいくつかぶら下がるという再帰的な構造であり，出発点は根とよぶノード１つである．多くのアルゴリズムで使われる重要なデータ構造である．

2.1 グラフ

印刷媒体ではノードを点で，枝を点をつなぐ線分で表すグラフ理論的な表示がよく使われ，階層構造が把握しやすい．それをそのま

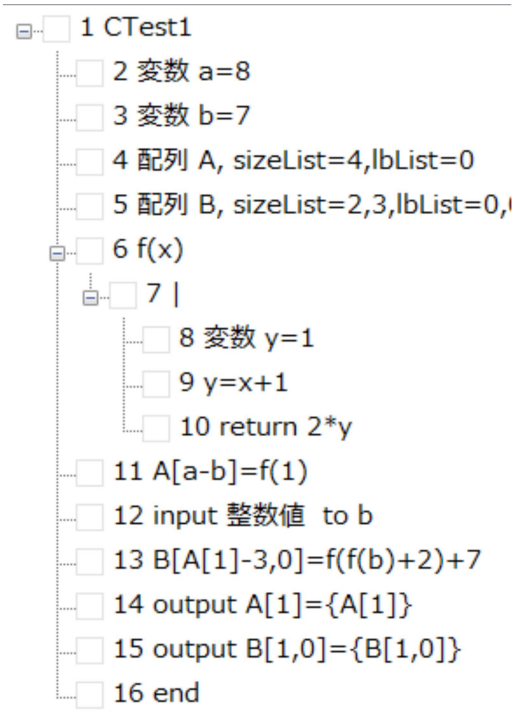
まコンピュータ画面で表示することもある．ただ，どうしても画面のスペースの制約を越えることは難しい．その困難を回避するため，本システムではTreeViewに着目する．これは多くのOSでファイルシステムの表示に使われ，見慣れているものである．

2.2 TreeView

これを用いて簡単なプログラムを表現した例を図1に示す．

1行1ノードであり，子ノードは親ノードより一段インデントを深くして並べる．

図１　TreeViewの表示例



ノード間の親子関係を示す線分が引かれている．また，記号⊟がついているノードがある．これをクリックすると，そのノードを根とする部分木が折りたたまれて根だけが表示され，そのことを示す記号⊞に代わる．逆に，⊞をクリックすると，部分木が展開される．また，ノードをクリックして選択するとハイライト表示される．このように展開折りたたみ機能は抽象度のレベルを変えてプログラムを見ることができるという意味があり，大きな利点となる．また，表示画面に入りきらないような大きな木ではスクロールバーが付き，見たい部分に移動してみることもできるので，大きな木でもとくに問題なく扱える．

2.3　インデントによる表現

図１のスクリーンキャプチャから枝を除き，記号⊟とかノードの一連番号を省いたテキストを考えていただきたい．そのような最小限の情報でもこの木構造を再現できるのである．「レベルをインデントの深さで表現して，括弧構造の括弧を省ける」と指摘したのが P.Landinである[1]．このアイデアはPython等で採用されよく知られるようになった．

2.4　LDR(Level-Data representation)

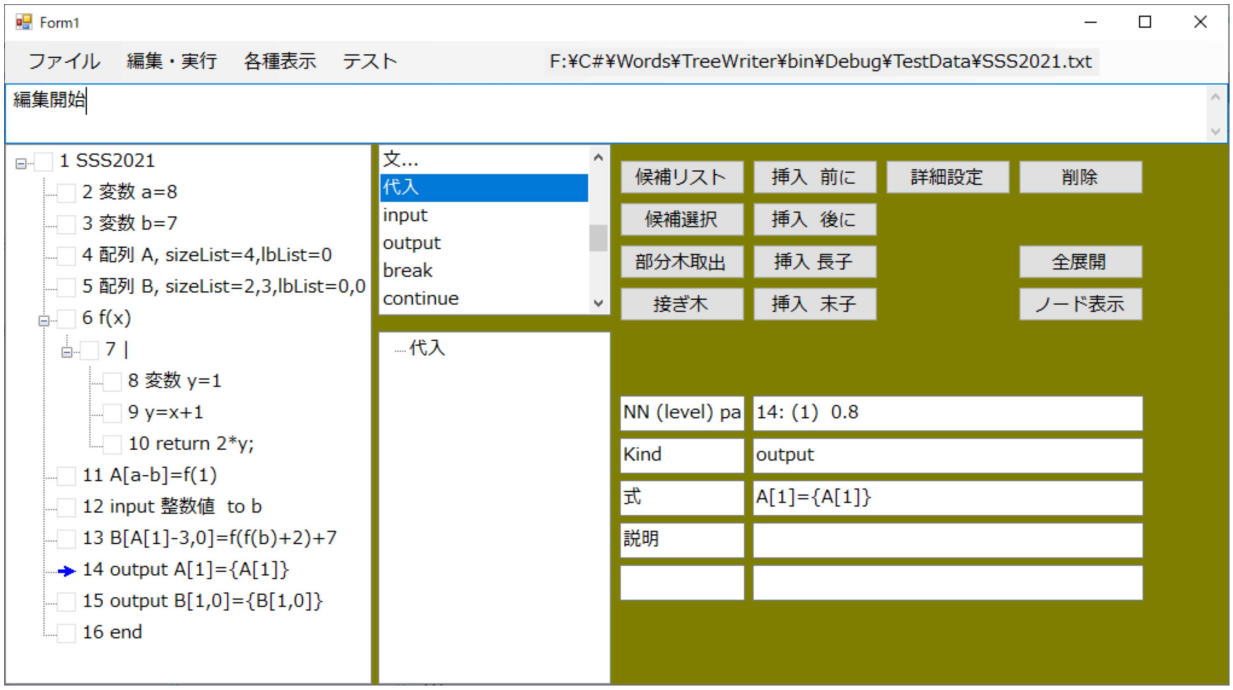
TreeView表現をファイルに保存し，また，逆にファイルから復元表示する必要がある．シリアル化と逆シリアル化である．本システムでは，インデントでなく，レベルの数とノード内容(をシリアル化したもの)を並べたテキストで表す **LDR**(Level-Data Representation)[3][4]を採用している．このシステムでの例として，図１の木をLDR表現したものの一部を付録として最後につける．

3.　木構造プログラミング

3.1 木構造プログラミングの発想

このシステムの発想は[2]に溯る．当時ソフトウェア工学で**保守**(maintenance)が大きな問題と認識されていた．我々はプログラムとドキュメントの乖離を縮めるために，抽象構文木としてプログラムを作り，必要なノードに説明を書き加える方法を提案した．木のノードは様々な抽象度レベルをもつが，それぞれのレベルで適切な説明を加えることを目指した．そして抽象構文木から具体構文のプログラムを生成する「代書系」の構想をもっていた．

図2　プログラムの入力編集画面



3.2　システムの画面

図２は試作システムのプログラム編集時の画面例である．上からメニュー，お知らせを表示するテキストボックス，その下は左(L)，中央(C),右(R)３つの領域に分割され，Lが編集中のTreeView(プログラム木)であり，中央Cは上下にペインC1,C2に分かれる．C1に挿入可能な構成要素のリストを表示する．その1つを選択すると，その木構造がC2に表示される．Rの上ペインR1には編集用のボタンや，実行用のボタンがある．Rの下ペインR2には，選択中のノードの詳細情報を表示する．これ以外に必要により種々のフォームを表示する．たとえば，入力用の小さいフォームにプロンプトを表示しテキストボックスで入力を受け付ける．

3.3　木構造プログラミングの入力・編集

ノードを1つずつ木に加えるという方法でもプログラムを作れるが，言語機能ごとに部分木を追加する形で編集する方が楽であるし，機能中心・概念中心の作業になる．編集の方法を説明する．

たとえば，図２の左ペーンLのプログラム木をみると，青い矢印が１つのノードを指しているが，この14番のノードが選択されていて，そのノードの内部情報の一部がR2ペーンに表示されている．

その一番上のエントリは３つの情報をまとめて表示している．NNはノード番号，（）内には,ノードのレベル，そして，最後の0.8は「ノードアドレス」である．

Kindはノードの種別で，この場合はoutputである．式というエントリに，出力させたいテキストを指定する．

この指定は文字列補間(string interpolation)を使える．{} の外は「地の文」で，{} の中に式が書いてあれば，それを評価した結果を文字列として埋め込んで，全体の文字列を作り上げてくれる．その方式をまねして，括弧の中にエントリ名を書くと，そのエントリの値を括弧のところに埋めて文字列を完成するようにしている．こうして，初学者に負担になりがちな出力文を簡単に指定出来る．

C1には用意してある機能要素のうちそのコンテクストで必要なものの一覧を表示し，その１つをクリックすると，C2にそのテンプレートとしての部分木が表示される．この例では代入なので部分木と言ってもノード１つでしかないが，たとえば，if2を選択するとtrue節，false節もついてくる．

いま14番のノードが選択され，C2には代入があるので，ボタンが沢山ある中から，**挿入 前に**をクリックすると,13と14の間に代入ノードが挿入される．**挿入 後**とかレベルを下げて子供のノードを**挿入 長子，挿入 末子**で入れることもできる．また，**接ぎ木**とあるのは，そのノードを部分木の根で置き換える，つまり，その位置に部分木をおくことを意味する．

大きさにかかわらず任意の部分木を部分木の根を選択し，**部分木抽出**をクリックすると，C2に部分木のコピーができる．これを任意の場所に接ぎ木できるので，任意規模の部分木のコピーが簡単にできる．

木のノードを挿入しただけでは名前も式も入っていないので，R2のエントリの必要なところを書き込み，**詳細設定**をクリックすると，それらのデータが内部に取り込まれる．詳細設定のクリックを忘れて他のノードにうつると折角入力した情報は消えてしまう（という実現になっているが，必要なら改良する）．

ノード表示は，Lのノードの表示方式を巡回式に変更する．ノードの前の番号を見せない表示とか,種々の表示を切り替えできる．

これらの操作を繰り返してプログラムの構造と詳細情報を設定し終われば，一度メニュの**実行準備**をクリックすると，ノード番号を付け直したりして見やすくなる．その段階で，**保存**をクリックすると編集結果を残せる．

以上のように，テキスト方式のエディタでも本質は変わらないのではあるが，この編集系を使っていると，部分木ごとに編集することの有り難みはときに感じることがある．

3.4 式と制御構造の分離

while(E) do S のように本来，式と実行文は複合している．式はこれを構文解析し，構文木として扱うこともできる．そうすれば，プログラム全体を抽象構文木で扱うという意味で一貫性があるが，式の木を作ってみるとかなり大きくなることがあり，逆に大局が見えにくいことにもなる．そこで，whileを代表するノードを造り，その中に式エントリを設けて，式は１つの文字列として扱う．実行部はwhileノードの子ノードに置く．このように制御構造と式は扱いを変えており，**分離方式**と称している．式は人類が長年月を掛けて洗練してきた表現様式であり，コンパクトに内容を表現する手段である．式に比べ実行制御の文の表現様式はまだ歴史が浅いということはあろう．

４．インタプリタ

4.1 式，制御構造，変数

編集しただけでは実行出来ない．**実行準備**メニューをクリックして，（コンパイラほどではないが）準備を行う．式を逆ポーランド記法に変換することと，変数の準備と変数表の作成等である．変数は１つずつノードを用いる．たとえば，変数ｘと宣言するとノードｘが作られ，その中にその値を保持できる．リスト，配列もそれぞれの名前のノードが作られ，その中にリスト，配列の本体を用意し値を保持する．([5][7]では，配列を要素に展開して部分木に持つようにしたが，実際にいくつかのプログラム例を実行してみて，すこしバランスが悪いと感じるし，表示も必ずしも見やすくないと判断し，ひとつの配列データは１つのノードにもつように変更した).

4.2 実行

ステップ実行と可変速の連続実行を用意している．実行中のノードは矢印で表示する．

inputは小さい入力専用のウィンドウを開いて入力を待つ．output出力はC１ペインに順次書き加えていく．またすべての変数の内容をC2ペーンに集約して表示している．

5　代書機能

文学では「シラノ・ド・ベルジュラック」など**代書**の例が見られるが，依頼者の想いを効果的な文書に仕上げることである．ここでは，抽象構文の木プログラムから具体構文に従うプログラムを生成することを代書と呼ぶ．プログラム生成という語もあるが，もっと抽象度の高い記述が出発点になる場合もありそうで，ここでは避けた．

具体的に先に図１でみたものをC#のプログラムに変換する例をみていただく．

図１の代わりに実質同内容のテキスト型式の図３と対比させてみていただく．図３ではレベルが[]の中に書かれている．

図3　レベルを括弧の中に示す

```
 1 [0]  CTest1
 2 [1]      変数 a=8
 3 [1]      変数 b=7
 4 [1]      配列 A, sizeList=4
 5 [1]      配列 B, sizeList=2,3
 6 [1]      f(x)
 7 [2]          |
 8 [3]              変数 y=1
 9 [3]              y=x+1
10 [3]              return 2*y

11 [1]      A[a-b]=f(1)
12 [1]      input 整数値   to b

13 [1]      B[A[1]-3,0]=f(f(b)+2)+7
14 [1]      output A[1]={A[1]}
15 [1]      output B[1,0]={B[1,0]}
16 [1]      end
```

→

図４　代書系の出力　C#プログラム

```
//Program CTest1
int a= 8;
int b= 7;
int[] A=new int[4];
int[,] B=new int[2,3];
int f(int x)
{
    int y;
    y=x+1;
    return 2*y;
}
A[a-b]=f(1);
Console.WriteLine($"整数値");
b=int.Parse(Console.ReadLine());
B[A[1]-3,0]=f(f(b)+2)+7;
Console.WriteLine($"A[1]={A[1]}");
Console.WriteLine($"B[1,0]={B[1,0]}");
Console.ReadKey();
```

5.1 例

図１を木プログラムを本システムの代書系で変換したのが図４である．行ごとに変換しているので，どうやっているかはほぼ理解できるのではないだろうか．

ただ，このままでは実行出来ない．Visual StudioでC#のコンソールアプリのプロジェクトを１つ作ると，図５のようなテンプレートが作られる．上の出力を図５の長方形の部分に貼り付けてコンソールアプリのできあがりとなる．ただ，貼り付けてコンパイルするとシンタクスエラーを指摘されることがある．たとえば，外部表現の“==”を内部処理では“≡”としており，それをそのまま出力してC#でエラーとされた．トークンレベルの置き替えは変換時に同時に行うこととした．

図４の例はとくに問題なく実行できて，コンソール画面は図６のようになった．

図５　コンソールアプリの額縁

```
using System;
namespace TestPro4
{
    class Program
    {
        static void Main(string[] args)
        {
            ここに，はめ込む
        }
    }
}
```

図６　実行例	説明
整数値	プロンプト
7	ユーザは７と入力
A[1]=4	計算結果
B[1,0]=45	

図４の代書例をみると，いくつか気づかれることがあろう．単純変数a,bは勝手にint型として出力している．これは当初お断りしたようにアルゴリズム教育を主眼としており，その際，型はあまり考慮しないですませており，実言語でのプログラムも普通はint（整数型）を使っているという事情による．C#は型に厳格な言語であり，指定を省略することは許されない．そのつぎもintの配列にしているが，配列のサイズだけを指定すればよいので，A,Bはそのサイズ情報だけを利用している．そのつぎは関数ｆであるが，これはint→intというシグニチャをもつメソッドとして扱われることになる．

引数はint xで，メソッド本体はint 変数ｙの宣言と,代入文とreturn文からなる．これは直訳で済んでいる．関数のつぎの代入はそのままの形でよい．ただし，お気づきと思うが，すべての文には終端子 ；が付加されている．inputはC#には直接対応するものがなく，プロンプトを出力し，入力を読み込んでこれをintに変換して指定された変数に代入している．

２つの出力文はConsole.WriteLineの中に，outputの式をそのまま書きだしている．最後の end は，意味的には全く対応しないConsole.ReadKey(); に置き換えているのが異質かもしれない．代書のルールをどう定めるかは検討課題である．

5.2　代書プログラムの変換法

ここでは，ノードの種類ごとの変換規則を定めることで変換できる範囲に限定している．いわば，直訳レベルに限定しており､高度の意訳的な変換は想定していない．

ルートから始めて再帰的につぎに示す変換ルールで，テキストを出力していく．インデント量はレベル情報を利用する．

5.3　変換ルールの例

ここでは，C#のコンソールアプリを作ることを想定しているので，それに合うようにルールを作る必要がある．

各構成要素ごとに書き換えのルールを作るが，それをつぎの図７にまとめる．

出力指定は多くが，{} を含む補間文字列である．たとえば，代入は{左辺}={式};としているが，代書系はこの記述から，該当ノードの左辺エントリから，左辺の指定を{左辺}に代入し，また，式エントリからそこに指定されたものを {式} に代入して，代入文を作成して,代書の出力とするのである．オペレータはC#と違うものがあるが，それを除けばごく標準の数式を受け入れており,式の構造を変える必要はない．

配列のサイズは図１のようにsizeListエントリに４とか２，３というように，次元ごとのサイズをリストとして表現している．４は大きさ４の１次元配列，2,3は２×３の２次元配列である．こういう表記を使うことで，３次元以上でも同様の記法で配列を扱える．このsizelistの指定法は期待以上に便利であっ

図７　変換ルール	
kind	出力文字列
program	//Program {名}
関数	int{名}(int {引数})
単純変数	int{名}={値}
配列	int[] {名}=new int[{sizeList}];
代入	{左辺}={式};
input	Console.WriteLine($"{prompt}"); {dst}=int.Parse(Console.ReadLine());
output	Console.WriteLine($"{式}");
break	break;
continue	continue;
return	return {式}
end	Console.ReadKey();
if1,if2	if({式})
while	while({式})
body true 節	{ } で括る
false 節	else{ }で括る

たし，期せずして，C#に非常にしっくりと対応できた．図１にはLbListというのもあり，０と0,0を指定している．これはindexの下限値を指定するもので，C系統の言語では無条件に０始まりである．ここではPascalなどの言語の存在も意識して，0始まり以外の配列も扱う積もりでこの指定を加えたが，その実現は後回しになっている．

ifやwhile等は判定式をif()，while()のカッコ内に入れれば変換でき，あとの処理はtrue節やelse節，あるいはループ本体(body)として変換する．

さて，代書の仕組みとルールを述べたが，これをみれば，C#以外の言語への対応も難しくはないと思われたのではないだろうか．

5.4　代書系の実現

図７の変換ルールは内部的には，辞書として用意し，kindをもとにその変換表から変換ルールを取り出して適用する．表を差し替えることで，ひとつの木プログラムからいろいろの対象言語へ代書できる．

C#への代書はかなり直接的であったが，他の言語についてはどうだろうか．最近の言語は似たり寄ったりという印象もあるが，複数の言語への代書法を検討することで，プログラム言語の個性を探るのにも使えるし，この代書法の限界も見えるであろう．

5.5　代書による教育の可能性

木構造でのプログラムシステムは，アルゴリズムの学習の際，実言語でのプログラミングという大きな負荷が学習者に掛かるのを避け，アルゴリズムそのものの学習に集中することを目指している．それでアルゴリズムについて十分学習し，適当なときに代書系を使って,実言語への橋渡しをすればよい．

今回のC#のコンソールアプリへの代書はどうだろうか．図５にコンソールアプリの額縁を見ていただいたが，なぜこう書くのか説明は難しい．説明できないので，これは「おまじない」という指導になる．額縁として扱い，代書した部分をはめ込む方式は学習者にどう受け取られるだろうか．

木プログラム方式でじっくりアルゴリズムを検討して，仕上げとして,機械的に代書し，実言語に移行するときに起こる諸々のことを体験するのはどうだろうか．機械的代書で得られたプログラムより，もっといい書き方を見つけることで実用言語の表現力の豊富さをあじわうこともあろう．

実用言語に変換することのメリットの１つは，リアルな環境での処理時間などの感覚が得られることである．

また，実言語で動けば，課題の仕様をすこし変更して,実言語プログラムでの改変を求めることで，徐々に実言語でのプログラミングになじむような方法も取れるであろう．言語の学習は文法書を一から読むより，具体例をみて，すこし変更してみて試してみるのが速いであろう．

６．むすび

木プログラムをつくる．ブロックプログラミングと同様，機能単位ごとに部分木を組み合わせていくので，テキスト指向から機能指向・概念指向に転換している．

型など，アルゴリズム学習に直接必要のないことは捨象された抽象的な世界を提供できる．そこで十分アルゴリズムを検討し，それから代書して実言語の記述を入手し，実行してみる．これは実環境での実行の手応えを与えてくれる．こうして，木構造でのアルゴリズム学習から実言語への橋渡しを意図している．

なお，アルゴリズム教育向きのシステムとしては[8][9]等いくつかの先行研究がある．代書方式を採用するものとしては，[10][11]の例が興味深い．[10]はブロック形のプログラムから，Javascript,Python,PHP等のプログラムを生成する．[11]ではビジュアルプログラミングシステムとJava言語の双方向の変換について詳細な検討が行われている．

謝辞　最後に本報告発表までに種々の支援をいただいた方々，有用なコメントを頂いた査読者お二人にも感謝申しあげます．

（連絡先：　nobtokura@gmail.com）

参考文献

[1]P.J.Landin:”The next 700 programming languages”，C.ACM 9(3), 157–166 (1966.03).

[2]Y.Nakamoto, M.Iwamoto, M.Hori, K.Hagihara and N.Tokura: An Editor for Documentation in Π–system to Support Software Development and Maintenance,6th In'l Conf. on Software Eng. 330–339, (1982.09).

[3]都倉信樹:　入門プログラミング教育につづく科目案 SSS2019, 6–1, (2019.08).

[4]都倉信樹：ある単純な木表現法の応用について，FIT2019，6D-7.(2019.09).

[5]都倉信樹: 木構造プログラミングシステムの試作，情処83回全国大会,5G-07,(2021.03).

[6]兼宗進，並木美太郎，長慎也，長瀧寛之，長島和平，本多佑希ほか：プログラミング事例集,高等学校教授用資料，東京書籍，（2021.04）．https://ten.tokyo-shoseki.co.jp/detail/114385/

[7]都倉信樹: 木構造プログラミングシステム,情処CE研160-6 (2021.06).

[8] 本多佑希，兼宗進: ブラウザ上で動作するDNCL学習環境「どんくり」の開発．CE研，1-4．(2018.10)

[9] 西田知博，原田章，中村亮太，宮本友介，松浦敏雄: 初学者用プログラミング学習環境 PEN の実装と評価.情処論,48(8)，2736-2747．(2007.08)

[10]Google Blockly
https://developers.google.com/blockly

[11]松澤芳昭，保井元，杉浦学，酒井三四郎："ビジュアル-Java相互変換によるシームレスな言語移行を指向したプログラミング学習環境の提案と評価",情処論,55(1),57-71，(2014.01)

付録　図Iの木をあるLDRで表現したもの.　ノード７から10の部分

〖Level⁞2〗 〖NN⁞7〗 〖■ ⁞f1〗 〖Kind⁞body〗
〖Level⁞3〗 〖NN⁞8〗 〖□ ⁞v〗 〖Kind⁞ 単純変数〗 〖名 ⁞y〗 〖値 ⁞1〗 〖説明 ⁞ 〗
〖Level⁞3〗 〖NN⁞9〗 〖□ ⁞s〗 〖Kind⁞ 代入〗 〖左辺 ⁞y〗 〖式 ⁞x+1〗 〖説明 ⁞ 〗 〖宛先 ⁞y〗
〖Level⁞3〗 〖NN⁞10〗 〖□ ⁞s〗 〖Kind⁞return〗 〖式 ⁞2*y〗 〖説明 ⁞ 〗 〖宛先 ⁞Fv〗

注：LDR はレベルを表す数と内容の文字列で構成することが基本である.ここでの例は，1 ノード一行で,最初にレベル,次にノード番号などが,キーと,その値の対として括弧で括って並べられている形を使っている．　ユーザには見えない情報も含まれている.