計算機プログラミング I (11)

再帰的なデータ構造

2003 年 1 月 9 日　増原 英彦

これまでに学んだデータの種類 (データ型) には

- 整数 (int) や実数 (double) のような基本型、
- 同じ型のデータを一列に並べた配列型と、
- 複素数のようにいくつかのデータ型を組み合わせたオブジェクト型

があった。実用的なプログラムでは、複数の値をまとめて一つの値として扱うことができる配列型やオブジェクト型が必須となる。

オブジェクト型は、基本型だけでなくオブジェクト型の値をも組み合わせることができる。つまり、オブジェクトのインスタンス変数には、基本型の値に限らず、他のオブジェクトをしまうことができる。(例えばタートルオブジェクトのインスタンス変数には、「X 座標」や「方向」といった数値だけでなく、「色」や「表示されている TurtleFrame」といったオブジェクト (の参照) がしまわれている。)

多くのプログラムには、値を一列に並べたようなデータ (cf. カラオケの予約リスト) や、値をグループ化し、それをさらにグループ化したようなデータ (cf. 組織構造や階層ファイル構造) のようなデータが現われる。このようなデータの設計をデータ構造という。

データ構造の中には、構造の定義に自分自身の定義を再帰的に使っているものがある。つまり、あるクラスのあるインスタンス変数が、そのクラス自身の型になるような設計をしている。このようなデータ構造を再帰的なデータ構造という。再帰的なデータ構造を使うと、無限の大きさを持つデータを簡単に表現できる。

1　リスト構造

リスト構造は、データを鎖のようにつなぎ、一列に並んだデータ表わすための再帰的データ構造である。配列と異なり、列の長さを増減させることや、列の途中にデータを挿入したり、取り除くことが簡単にできる特徴がある。

以下では、マウスボタンをクリックによって、画面上のタートルを増減させることのできるプログラム ListDemo を例にリスト構造の定義と使い方を紹介する。ListDemo では画面上にあるタートルを管理するためにリスト構造 TurtleList を用いる。リストには、タートルが何匹か登録されており、マウスクリックによってタートルが追加されたり、取り除かれたりする。また、リストに登録されたタートルを全て前進・回転させる機能もある。

1.1　リスト構造の定義

リスト構造は、細かい輪がつながった鎖に例えることができる。鎖を作るの 1 つ 1 つの輪はリスト要素とよばれ、その輪にぶら下がっている「先頭のデータ」と隣の輪に相当する「次のリスト要素」を覚えている。Java 言語のクラスでは次のように定義できる:

```
1 public class TurtleList {
2     public Turtle     firstTurtle;              // 先頭のタートル
3     public TurtleList nextTurtles;              // 続くタートルのリスト
4     public TurtleList(Turtle f, TurtleList n) { // コンストラクタ
5         firstTurtle = f;
6         nextTurtles = n;
7     }
8 }
```

インスタンス変数 nextTurtles は TurtleList のオブジェクトをしまうことができる。つまり、 TurtleList クラスは、定義中に TurtleList クラス自身が使われている再帰的なデータ構造になっている。

1.2 リストを作る

タートルのリストを使うプログラム (ListDemo) は、ボタンが押されるのを待ち、押されたボタンに応じた処理をする。まず「追加」というボタンが押されると、タートルを 1 つ作り、それをリストに追加する部分を見る:

```
1  public class ListDemo {
2      public static void main(String[] args) {
3          ウィンドウを作る (略)
4          TurtleList l = null;                    // タートルのリスト; 始めは空
5          while (true) {
6              if (「追加」ボタンが押された) {
7                  int x = (略), y = (略);          //クリックされた場所を調べる
8                  Turtle m = new Turtle(x,y,0);   //その場所にタートルを作り
9                  m を画面に表示する (略)
10                 l = new TurtleList(m,l);        //リストにタートルを追加する
11             } else
12                 他のボタンが押されたときの処理 (略)
13         }
14     }
15 }
```

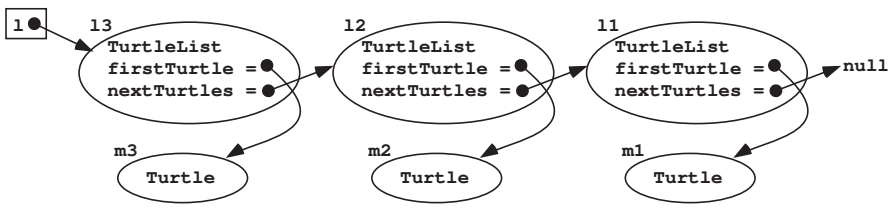
このプログラムでは変数 l にタートルのリストをしまっている。4 行目にあるように、最初、リストは「空」の状態である。null は「どのオブジェクトも参照していない」ことを示す特別な値である。

リストを作る典型的な方法は、すでにあるリストの先頭に「リスト要素」を 1 つずつ追加してゆくものである。10 行目では、新しく作られたタートルを「先頭のタートル」、すでにあったリストを「次のリスト要素」とするリスト要素を作り、それを l に代入している。結果としてすでにあったリストの先頭に要素を 1 つ追加したことになる。

例として、「追加」ボタンが 3 回押され、タートルが 3 匹追加される場合を考えてみよう。

1. 最初、l は null, つまりリストは「空」である。
2. 「追加」が押されると、タートルが 1 つ作られ (m1 とする)、firstTurtle が m1 で、nextTurtels が l, つまり null であるようなリスト要素が作られる (これを l1 とする)。この時点で l は m1 だけからなる長さ 1 のリストである。
3. もう 1 度「追加」が押されると、タートルがもう 1 つ作られ (m2 とする)、firstTurtle が m2 で、nextTurtels が l, つまり l1 であるようなリスト要素が作られる (これを l2 とする)。この時点で l は m2, m1 からなる長さ 2 のリストになる。
4. さらに「追加」が押されると、firstTurtle が新しいタートル (m3) で、nextTurtels が l, つまり l2 であるようなリスト要素 (l3) が作られる。

最後の状態を図示すると次のようになる:



1.3 リストに対する繰り返し

「前進」ボタンが押されると、リストに含まれている全てのタートルが 10 だけ前進する。これを実現するには、リスト中の全ての firstTurtle に対して fd(10) というメソッドを呼び出せばよい。再帰的なデータ構造の全ての要素に対して処理をしたり計算をするには、再帰的メソッド呼出を使った定義が適している。この場合、つまり リストに含まれる全てのタートルを前進させる には

まず、先頭のタートルを前進させ、
次に、続く リストに含まれている全てのタートルを前進させる

と考えることができる。これをプログラムにすると、次のようなインスタンスメソッドを TurtleList クラス に追加することになる:

```
1  public class TurtleList {
2      インスタンス変数・コンストラクタの定義（略）
3      public void forwardAll(int s) {              // 全てのタートルを s だけ前進
4          firstTurtle.fd(s);                       // 先頭のタートルを前進
5          if (nextTurtles != null) {               // もし続くリストがあれば
6              nextTurtles.forwardAll(s);           // 続くリストの全てのタートルを前進
7          }
8      }
9  }
```

6 行目は「続くリスト要素」(nextTurtles) 中の全てのタートルを前進させる再帰的なメソッド呼び出しである。現在のリスト要素がリストの最後だった場合、nextTurtles が null になっているので、5 行目の if 文を使って nextTurtles が null でない場合のみメソッド呼び出しを行う。

練習 11-1: (タートルの回転)　リストに含まれる全てのタートルを回転させるインスタンスメソッド void turnAll() を TurtleList クラスに定義し、さらに ListDemo クラスを変更して「回転」ボタンがクリックされるとこのメソッドを呼び出すようにせよ。回転する角度は 0 ~ 360 の乱数で決めることとする。(ヒント: Math.random() は 0 以上 1 未満の実数 (double) 型の乱数を発生させる。)

練習 11-2: (リストの長さ)*　リストの長さを数えるインスタンスメソッド int length() を TurtleList クラスに定義せよ。(ヒント: リストの長さは「続くリストの長さ」+1 である。) ListDemo を変更して、追加や削除がされる度に、現在のリストの長さを表示させよ。(注意: リストが空 (null) のときは、メソッド呼び出しはエラーになるので、リストが空でない場合にのみ表示するようにしなければいけない。)

練習 11-3: (最上位置のタートル)　リストに含まれるタートルのうち、Y 座標が最も小さい (つまり画面の一番上の位置にある) ものを探すインスタンスメソッド Turtle findTop() を TurtleList クラスに定義せよ。さらに ListDemo を変更して「移動」ボタンによって全タートルが移動したら、最上位置にいるタートルの色を赤く変えるようにせよ。

ヒント: あるリスト要素 (から始まるリスト中) で「最上のタートル」は、「続くリスト要素中で最上のタートル」と、「先頭のタートル」のうち、より上方のものになるので、次のようになる:

```
1  public class TurtleList {
2      インスタンス変数・コンストラクタ・他のメソッド（略）
3      public Turtle findTop() {
4          if (nextTurtles == null)
5              return firstTurtle;          //続くリスト要素がない場合は先頭のタートルが最上位置
6          else {
7              Turtle topOfNext = 続くリスト要素中で最上位置のタートルを探す;
8              firstTurtle と topOfNext の Y 座標を比較し、小さい方を return する
9          }
10     }
11 }
```

また、タートル m の色を変えるには、`m.kameColor = java.awt.Color.red; m.fd(0);` のようにする。(`m.fd(0);` は画面に表示されているタートルの色を即座に変えるためのトリックである。)

1.4 リストの最後に要素を追加する

1.2 で見たように、すでにあるリストの 先頭に 要素を追加するには、いままでのリストを「続くリスト要素」とするようなリスト要素を作ればよかった。一方、リストの 最後に 要素を追加するには、リストの最後の要素を探し、それの「続くリスト要素」に新しいリスト要素をしまえばよい。そのためには次のような再帰的なメソッドを TurtleList クラスに追加すればよい:

```
1  public class TurtleList {
2      インスタンス変数・コンストラクタ・他のメソッド（略）
3      public void append(Turtle m) {
4          if (nextTurtles == null) {                // 自分がリストの最後なら
5              nextTurtles = new TurtleList(m,null); // 続くリスト要素を追加
6          } else {
7              nextTurtles.append(m);                // 続くリスト要素に m を追加
8          }
9      }
10 }
```

ListDemo の側では、新しく作ったタートルを引数としてこの append メソッドを呼び出すことでリストの最後にタートルを追加できる:

```
1  if (「追加」ボタンが押された) {
2      int x = (略), y = (略);                 //クリックされた場所を調べる
3      Turtle m = new Turtle(x,y,0);        //その場所にタートルを作り
4      m を画面に表示する（略）
5      l.append(m);                         //リストにタートルを追加する
6  } else
7      他のボタンが押されたときの処理（略）
8
```

1.5 リストから要素を削除する

「削除」ボタンが押されると、一番最後にリストに追加されたタートルがリストから取り除かれる。これを実現するには、変数 l が参照している「先頭のリスト要素」を、「続くリスト要素」に書き換えるだけである。

1.2 節の図では l は l3 から始まるリストを参照していたが、l を l3.nextTurtles の値、つまり l2 への参照に書き換えると、m3 を含む先頭が「消された」リストを参照することになる。

ListDemo 中では、「削除」が押されたときに l を書き換えればよいので、次のようになる:

```
1  if (「追加」ボタンが押された) {
2      タートルを追加する
3  } else if (「削除」ボタンが押された)) {
4      l = l.nextTurtles;              // リストの先頭を「続くリスト」に変える
5  } else
6      他のボタンが押されたときの処理（略）
```

(注: この例ではタートルは画面に表示されたままになる。ただし、「前進」を押してもリストから外れたタートルは前に進まなくなる。)

次に、画面上で 1 番上の位置にあるタートルを 1 つリストから削除することを考える。

この場合、リストの途中にある要素を取り除くことになるが、これには、その要素の 1 つ前の「続くリスト要素」を書き換えることになる。例えば、1.2 節の図にあるリストから m2 を含む要素を取り除くには、1 つ前、つまり l3 の nextTurtles を、(現在 l2 である状態から) l2 の nextTurtles が参照している l1 に書き換える。

リストの先頭のタートルが画面上で 1 番上にあった場合も考えると、リストから特定のタートルを含む要素を取り除くメソッドは、次のように「要素が取り除かれたリストを 返す メソッド」として定義するのがよい:

```
1  public class TurtleList {
2      インスタンス変数・コンストラクタの定義・他のメソッドの定義（略）
3      public TurtleList delete(Turtle m) {          // m を含むリスト要素を除いたリストを返す
4          if (firstTurtle == m) {                   // m を含んでいたら
5              return nextTurtles;                   // 続くリスト要素を返す
6          } else {                                  // m を含んでいなかったら
7              nextTurtles = nextTurtles.delete(m); // 続くリスト要素から m を除いて
8              return this;                          // 自身を返す
9          }
10     }
11 }
```

7 行目では「続くリスト要素」(nextTurtles) の delete メソッドを再帰的に呼び出し、その返り値を nextTurtles に代入し直していることに注意せよ。もし「続くリスト要素」に m があった場合、「続くリスト要素」の次のリスト要素が返り値となるので、結果として次の要素が削除される。

ListDemo でも、下の 5 行目のように、「l から m が取り除かれたリスト」を l に代入し直す。もし m が l の先頭にあった場合、l の値も変更しなければならない。一方、m が l の先頭にない場合は、最初のリスト要素は同じオブジェクトである。下のように l.delete(m) の返り値を l に代入し直すことで、両方の場合に正しく働く:

```
1  if (「追加」ボタンが押された) {
2      タートルを追加する
3  } else if (「削除」ボタンが押された)) {
4      Turtle m = リスト中で一番上にあるタートルを探す;
5      l = l.delete(m);                                    //m をリストから外す
6  } else
7      他のボタンが押されたときの処理（略）
8
```

練習 **11-4:** (最上のタートルの削除)　ListDemo の中の削除ボタンが押されたときの定義を変更し、画面の一番上の位置にあるタートルをリストから外すようにせよ。(画面の一番上の位置にあるタートルは、練習 11-3 で定義したメソッドを使う。)

練習 **11-5:** (特定の場所への追加)*　ListDemo のリスト中に、タートルが X 座標の小さい順に並ぶように、タートルを追加するときにリストの途中にに追加するようにせよ。タートルが移動して X 座標が変化する場合は考えないことにする。(ヒント: タートルをリストの適当な位置に追加し、追加されたリストを返すメソッドを定義すればよい。このメソッドは、自身の 手前に 新しいタートルを挿入すべきかどうかを判断し、挿入する場合は、自身を「続くリスト要素」とするようなリスト要素を作り、返せばよい。)

練習 **11-6:** (配列との比較)　ListDemo と同じ機能を持ち、リスト構造を使わずに、配列を使って「全てのタートル」を管理するようなプログラムを作ってみよ。配列では簡単に行えない操作は何か。

第 2 回課題: 描画エディタ

- 期限: 1 月 22 日 (水)
- 提出物と提出方法: a ~ e の各項目について
  - コンパイル可能なプログラムファイルをスクリプト実行によって提出
  - 課題の考察および解いた方法の説明を以下のどちらかの方法で提出
    - HTML 形式で作ったファイルをスクリプト実行によって提出、あるいは
    - 紙に書かれたものをレポートボックスへ提出

  (提出スクリプトについては web ページの説明を参照せよ。)
- 注意事項:
  - 考察・説明を紙で提出する場合でも、プログラムはスクリプト実行によって提出すること。
  - プログラムは **1** つのクラスを **1** つのファイルとし、項目 **(a ~ e)** それぞれについて、どのファイルが対応しているかを示した索引ファイルと一緒に提出せよ。考察・説明を **HTML** ファイルで提出する場合は、その先頭に索引を書け。
  - 常識の範囲を越えた類似部分のあるレポートがあった場合は、追加の面接試験を行う場合や、当該レポートの評価を 0 点にすることがある。
  - 全ての項目を提出しなくてもよい。
  - 複数の項目をまとめた場合には、どの項目をどのようにまとめているかが分かるように明記すること。明記されていない場合は、全て提出されていないと見做される場合がある。
  - 提出物の中心は考察および解いた方法の説明である。(プログラムは説明が正しいことの証明に過ぎないので、それだけを提出してはならない。)
  - レポートの読みやすさ・独自性も採点の対象である。

マウスを使って円や直線を描くことができる描画エディタを以下の手順に従って作成せよ。

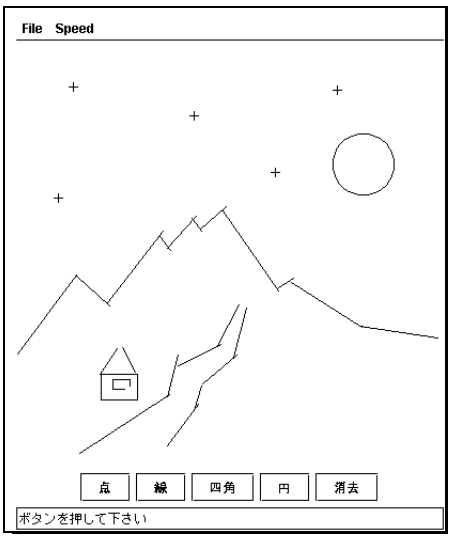


図 1: 描画エディタの画面例

a) 基本　描画エディタは、画面に描かれる円や線を表わす図形要素オブジェクトと、マウスからの入力を読みとり、画面表示を行うエディタ部分から成る。

円や線を表わすオブジェクトは void draw(Turtle m) というインスタンスメソッドを持つ。このメソッドが呼び出されると、タートル m を使って画面上に図形を描く。円や直線など形によって描き方が違うので、それぞれに対してクラスを定義する。描画エディタでは、後述するように「全ての図形要素を描き直す」ような

処理を行う。そのために、円や直線を表わすクラスは基本となる図形要素 (FigureElement) クラスの子クラスとして定義し、draw メソッドを再定義することで、図形を描き分ける。

共通の親クラスとして定義する FigureElement クラスは、次のように、何も描かない「何もない図形」になる[1]。

```
1  public class FigureElement {    // 何もない図形要素
2    public void draw(Turtle m) {  // m を使って画面に図形を描く
3        // 何もないので何も描かない
4    }
5  }
```

このクラスを親クラスとして、点や直線は次のように定義される:

```
1  public class Point extends FigureElement { // 点図形要素
2    int x, y;                                // 点の座標値
3    int size = 4;                            // 表示サイズ
4    Point(int x0, int y0) { x = x0; y = y0; }// コンストラクタ
5    public void draw(Turtle m) {             // 点を表示する (十字を描く)
6      m.up();   m.moveTo(x+size,y,0);        // 亀を点の少し下に移動
7      m.down(); m.fd(size*2);                // 上方向へ線を描く
8      m.up();   m.moveTo(x,y-size,90);       // 亀を点の少し左に移動
9      m.down(); m.fd(size*2);                // 右方向へ線を描く
10   }
11 }
```

```
1  public class Line extends FigureElement { // 線図形要素
2    int startx, starty, endx, endy;         // 始点・終点の座標値
3    Line(int x1, int y1, int x2, int y2) {  // コンストラクタ
4        startx = x1; starty = y1;
5        endx   = x2; endy   = y2;
6    }
7    public void draw(Turtle m) {            // 線を表示する
8      m.up();   m.moveTo(startx,starty,0);  // 始点に移動し
9      m.down(); m.moveTo(endx,  endy,  0);  // ペンを下ろして終点に移動
10   }
11 }
```

エディタ部分は、押されたマウスボタンに対応して図形要素オブジェクトを作り、その draw メソッドを呼び出して表示をする。概略は次のようになる:

```
1  public class Sketch {
2      public static void main(String[] args) {
3          ウィンドウを作る (略);
4          Turtle m = new Turtle(); // 描画用のタートルを作る
5          f.add(m);
6          while (true) {
7              if (「消去」ボタンが押された) {
8                  画面を消去する
9              } else {
10                 FigureElement e = null;          // 作られる図形要素をしまう変数
11                 if (「点」ボタンが押された) {
12                     int x = (略), y = (略);       // クリックされた場所を調べる
13                     e = new Point(x,y);          // 新しく点を作る
14                 } else if (「線」ボタンが押された) {
15                     int x1 = (略), y1 = (略);    // クリックされた場所を調べる
16                     int x2 = (略), y2 = (略);    // クリックされた場所を調べる
17                     e = new Line(x1,y1,x2,y2);   // 線を作る
18                 }
19                 e.draw(m);                       // 作った図形要素を表示する
20             }
21         }
22     }
23 }
```

[1] このようなクラスは実際にオブジェクトを作ることがないので、抽象 (abstract) クラスとして定義するのが一般的であるが、この授業では簡単のために、「何もしない」クラスとして定義している。

四角と円を表わす図形要素クラス Box, Circle をそれぞれ定義し、Sketch クラスと組み合わせて図形エディタを完成させよ。(注: 四角は直線の、円は点の子クラスとして定義することもできる。)
(参考プログラム Sketch.java, Point.java, Line.java)

b) 取消　以下の手順に従って、作図中に追加した図形要素を取り消すことができる SketchB を作れ。

- 図形要素のリスト構造、FigureList クラスを定義する。
- Sketch クラスを次のように変更した SketchB クラスを作る:
  - 描かれた全ての図形要素を覚えておく FigureList 型の変数 l を用意する。
  - 新たに図形要素が追加されたときには、画面に表示すると同時に l にも追加する
  - 「消去」ボタンが押されたときには、まず l から先頭の要素を取り除き、一度画面を消去し、l に残っている全ての図形要素をもう一度描く

c) 図形の削除　以下の説明に従ってマウスでクリックした図形を消去できる SketchC を作れ。

- 図形要素クラス (とその全ての子クラス) に double distance(int x, int y) というメソッドを追加し、点 (x, y) からの図形要素までの距離を計算できるようにする。(距離の定義は自由であるが、図形の輪郭線からの距離が直感的だろう。例えば中心が (u, v) で半径 r の円であれば、円周からの距離、つまり、$\left|r - \sqrt{(x-u)^2 + (y-v)^2}\right|$ を距離とするのが一つの定義である。)
- 「消去」ボタンが押されたときは、以下のような処理をする:
  1. マウスで画面上の点がクリックされるのを待つ
  2. クリックされた座標から最も近くにある図形要素をリストから探し、
  3. その図形をリストから取り除き
  4. 画面を消去して残った全ての図形要素をもう一度描く

d) 移動　以下の説明に従ってマウスでクリックした図形を移動できる SketchD を作れ。

- 図形要素クラスに void moveBy(int dx, int dy) というメソッドを追加し、図形の X,Y 座標を (dx, dy) だけ増やすようにする。
- 「移動」ボタンを追加し、ボタンが置されたときには、まず移動させたい図形要素をクリックさせ、次に移動させたい場所をクリックさせる。1 度目と 2 度目の場所のベクトル差 (dx,dy) を計算し、移動させる図形要素の moveBy メソッドを呼び出す。最後に、画面を消去して全ての図形要素をもう一度描き直す

e) 図形要素と機能の追加　描画エディタに (1) 星形や多角形のような図形要素や (2) 図形の回転・彩色のような機能を追加した SketchE を作れ。

図形要素には星形、家、木、折れ線・多角形などが考えられる。(ヒント: 折れ線は、頂点のリストをインスタンス変数として持つクラスとして定義できる。折れ線の形をマウスで指定させるには、頂点の位置を順にクリックさせ、原点あるいは最初の頂点の近くをクリックしたら全ての頂点を指定し終えた、とするのがよいだろう。)

図形の回転・彩色のためには、draw メソッドによって各図形を描き始める際に、最初に色や角度を設定する必要がある。また、回転のためには、形を座標値ではなく、角度と方向で覚えておいた方がよいので、図形要素クラスの設計をやり直す必要があるかも知れない。