計算機プログラミング I (7)

再帰

2000 年 11 月 21 日 増原 英彦

1 再帰

ある物事の定義に、その物事自身を使うようなやり方を、再帰的(recursive) 定義という。このような定義は数学・論理学の記述には多くみられる。プログラミングにおいても再帰的な定義ができると、より簡潔なプログラムを作ることができる。

1.1 再帰的定義の例

- $(a \times b)$ (b は整数) は、$(a \times (b-1))$ に a を加えたもの。ただし、$(a \times 0)$ は 0.
- x の n 乗 (x^n) は、x の $(n-1)$ 乗に x を掛けたもの ($x^{n-1} \cdot x$). ただし、x の 0 乗は 1.
- n の階乗 ($n!$) は、$(n-1)$ の階乗に n を掛けたもの ($n \cdot (n-1)!$). ただし、0 の階乗は 1.
- n 番目のフィボナッチ数 f_n は、$(n-1)$ 番目のフィボナッチ数と $(n-2)$ 番目のフィボナッチ数の和。ただし、0 番目,1 番目のフィボナッチ数は 1.
- 組み合わせ数 ${}_nC_m$ は $({}_{n-1}C_{m-1} + {}_{n-1}C_m)$. ただし、${}_nC_0$, ${}_nC_n$ はそれぞれ 1.
- a, b (ただし $a > b$) の最大公約数 $(GCD(a,b))$ は、a で b を割った余り r が 0 でなければ b, r の最大公約数 $(GCD(b,r))$, 0 ならば b.
- 命題論理の式 P の真偽値は、もし P が $Q \vee R$ という形をしていた場合、論理式 Q, R の真偽値を調べ、どちらか一方でも真であれば真 (他の形の式も同様)。もし P が "true" または "false" であれば、それぞれ真または偽、とする。
- n チームから成るトーナメント戦の勝者は、n チームを半分に分け、それぞれのトーナメントを戦行い、それぞれの勝者どうしを対戦の勝者である。ただし、1 チームから成るトーナメント戦の勝者は、その唯一のチームである。

1.2 再帰的なメソッド

再帰的に定義された計算は、ほぼそのままメソッドの本文中でそのメソッド自身を使うことで定義できる。例えば掛け算であれば下のようになる。この定義は、これまで見てきたメソッドとほとんど同じである。唯一違うのが、`times` というメソッドの中で`times` というメソッド自身を使っている点だけである。

```
1 // 掛け算の再帰的定義（足し算だけで定義している）
2 static double times(double a, int b) {
3   if (b == 0) return 0;          // a× 0=0          より
4   else return times(a,b+1) - a; // a× b=a× (b+1)-a より
5 }
```

注意すべき点としては、「基本の場合」(上であれば $b = 0$ の場合) を必ず書かなければいけないことがある。これがないと、永遠に繰り返しを続けてしまうことになる。また、再帰的にメソッドを使う間に「基本の場合」にたどりつく定義でなければいけない。例えば $a \times b = a \times (b + 1) - a$ だからといって下のように定義すると、やはり永遠に計算が続いてしまう。

```
1 // 掛け算の再帰的定義（足し算だけで定義している）
2 static double times(double a, int b) {
3   if (b == 0) return 0;          // a× 0=0          より
4   else return times(a,b+1) - a; // a× b=a× (b+1)-a より
5 }
```

練習 7-1: (再帰的メソッド)　上の例に出た $a \times b$, $n!$, n 番目のフィボナッチ数, ${}_nC_m$, a, b の最大公約数を求めるメソッドをそれぞれ定義してみよ。

1.3　変数の扱い

いままで変数は、「同じ名前でもメソッドごとに別のものが存在する」と説明してきたが、より正確には、メソッドが使われるたびに別のものが作られる。例えば、上の `times` にはa, b という 2 つの変数が存在する。いまメソッドmain の中で`times(5,2)` のように使われたとしよう。すると、

- 最初に使われた`times` のために変数a, b という箱が作られ、値は 5 と 2 になる。この中で`times(a,b-1)` のように使われる。
- 2 回目に使われた`times` のために変数a, b という箱が作られ、値は 5 と 1 になる。この中で`times(a,b-1)` のように使われる。
- 3 回目に使われた`times` のために変数a, b という箱が作られ、値は 5 と 0 になる。

のように、変数a, b のための箱が 3 個ずつ作られそれぞれが違った値を持つことになる。

2　組み合わせ探索

あらゆる組み合わせを作って試す場合、再帰的メソッドによる定義が便利なことが多い。例えば、覆面算や魔方陣のような問題を考えよう。これらの問題は n 個の変数に 1 桁の数を重複を許さずに割り当てるような組み合わせを作り、それが条件を満たしているかを調べるものと言える。

「n 個の変数に 1 桁の数を重複を許さずに割り当てた組み合わせ」を再帰的に定義すると次のようになる:

1. 「$(n-1)$ 個の変数に 1 桁の数を重複を許さずに割り当てた組み合わせ」をひとつ作り、$[x_1, x_2, \ldots, x_{n-1}]$ とする。
2. $x_n \notin \{x_1, x_2, \ldots, x_{n-1}\}$ なる x_n をひとつ選ぶ。

3. $[x_1, x_2, \ldots, x_{n-1}, x_n]$ は「n 個の変数に 1 桁の数を重複を許さずに割り当てた組み合わせ」の 1 つになる。

上の 1. と 2. で「ひとつ」選んでいたものを「全てに」ついて行えば、全ての組み合わせを作ることができる。

これをメソッドとして定義するには、ひとつずつ異なる組み合わせを作ってそれを処理するというスタイルにするのがよい。概略を示すと、n 個の要素から成る組み合わせ探索をするような場合は、下のようなプログラムの構造になる。

```
static void 組合せを作る(これまでに作られた組合せ) {
  if (「これまでに作られた組合せ」に n 個の要素がある、つまり、組み合わせが完成した場合)
    チェックするメソッド(これまでに作られた組合せ);
  else
    for (全ての組み合わせ要素 (x とする) について調べる)
      if (x を「これまでに作られた組合せ」に追加できる)
        組合せを作る(これまでに作られた組合せに x を追加したもの);
}
```

```
 1  // 全ての 1 桁の数を重複せずに並べたものを作る
 2  // combination(n, i) が呼ばれたとき、n[0]..n[i-1]
 3  // まではすでに値が決っているものとする。
 4  static void combination(int[] n, int i) {
 5    if (i == n.length) // 全ての桁が決った場合
 6      checkExpression(n);  // その数が条件を満たしているかを調べる
 7    else {
 8      // 次の 1 桁 (nextNumber) を 0 から 9 として計算を進める
 9      for (int nextNumber = 0; nextNumber < 10; nextNumber=nextNumber+1) {
10        // n[0]..n[i-1] で nextNumber が使われていなければ
11        if (!used(n,i,nextNumber)) {
12          n[i] = nextNumber;   // i 桁目にセットして
13          combination(n, i+1); // 次の桁以降を決める
14        }
15      }
16    }
17  }
```

図 1: 全ての組み合わせについて試すプログラム

「1 桁の数字の重複を許さない割り当て」の場合は、

- n 個の変数を整数の配列 (と、何番目まで値が決まったかを表わす数値) で表し、
- 「全ての組み合わせ要素 (x とする) について調べる」の部分には、0 ~ 9 について調べ、
- 「x を「これまでに作られた組合せ」に追加できる」部分には、x が「これまでに作られた組合せ」に含まれているかどうかの検査、
- 「これまでに作られた組合せに x を追加したもの」には配列への x の代入になる。

結局、図 1 のようなメソッドになる。6 行目で使われているメソッド checkExpression 中で、作られた組み合わせ n が条件を満たしているかを調べることで、覆面算や魔方陣を解くプログラムが完成する。また、11 行目で使われているメソッドused(n,i,nextNumber) は、配列 n の 0 から (i − 1) 番目までに nextNumber が現われるかどうかを調べるものである。(⇒ 覆面算の例については web ページ参照。)

練習 7-2: (魔方陣) 覆面算のプログラムを参考に魔方陣を解くプログラムを作れ。ただし、この魔方陣の大きさは 3×3 で、各行・各列・各ナナメの列の和が全て等しいものとして、使う数字に重複はないものとする。

3 第 2 回レポート課題

以下の 2 問両方についてプログラムを作成し、説明・実行結果とともに提出せよ。ただし、「(optional)」と書かれた小問は余力があるもののみ解けばよい。

提出方法: レポートは教育用計算機システムの電子メールによって以下の提出用アドレスに送ること。

提出用: masuhara-cp1-report2@lecture.ecc.u-tokyo.ac.jp
相談用: masuhara-cp1@lecture.ecc.u-tokyo.ac.jp

期限: 2000 年 12 月 2 日 (土) — 特別な事情がある場合を除いて、遅れて提出することは認めない。

配点: 全体の成績の 25 点分に相当する。

注意:

- レポート提出の際、cc:フィールドに自分のアドレスも書いて、自分にも同じメッセージを送っておくこと。(紛失した際の確認のため。)
- レポートは 1 つの電子メールメッセージにまとめること。添付ファイルはプレーンテキストとして読める形式であれば使ってもよいが、MIME base64 encoding 等の形式のものは無視するので注意せよ。(この意味が分からなければ、添付ファイルを使わない方がよいだろう。)
- 期限内にレポートを出し直したい場合は、レポート全体を再度送ればよい。逆に言うと、最後に送られたメールだけを見て採点するので、追加分だけを送らないこと。
- 説明や実行結果のないレポートは採点しない。プログラムおよびレポートは、必ず自分で作成すること。他人のレポート (プログラムを含む) を写したものがあった場合は、写した者、写された者の両者ともに 0 点とすることがある。
- 提出されたメールが正しく受け付けられた場合、数分以内に提出内容の全文を引用した確認のメッセージが折り返し送られるのでそれを必ず確認せよ。もし上手く受け付けられない/確認メッセージ来ない場合は相談用のアドレスに〆切前に相談すること。
- レポートに絵や画像を含めたい場合は、web ページ上に画像を用意し、レポートにはその URL のみを書くこと。

課題 2-1 (冪乗)

x の n 乗 (x^n) を計算するメソッドを再帰的に定義したい。

1. (必須) x の n 乗を計算するメソッドを定義し、それを使って以下の計算を行え:
   (a) 2 の 31 乗
   (b) 10 万円を金利 0.45% で 10 年間預けた後の残高、つまり、10 万円 $\times (1 + 0.45/100)^{10}$
   (c) 43 個の整数から無作偽に選ばれた 6 個を当てるゲームを考える。1 回のゲームで当たる確率を $r(= 1/{}_{43}C_6 = 1/6096454)$ としたとき、このゲームが 365 回連続して外れになる確率 $(1 - r)^{365}$
2. (optional) ある数の偶数乗を求める場合に、 $x^{2n} = (x^n)^2$ という関係を使うことを考える。x の n 乗を計算するメソッドを、この関係を用いて改良せよ。また、メソッドが使われる回数を $n = 100$ の場合で比べてみよ。
3. (optional) t 年後に元金が r 倍になるような利率を漸近解法によって求めるメソッドを定義して、10 年後および 100 年後に元金が 2 倍になる利率を計算せよ。(検算もせよ。)
   (※ r の t 乗根を計算していることになる。)

課題 2-2 (複素数の計算)

$(x + iy)$ のような複素数を、$a[0] = x,\ a[1] = y$ であるような大きさ 2 の`double` 型の配列 a で表わすことにする。

1. (必須) 以下のようなメソッドを定義し、与えられた計算の結果を表示させよ。

   メソッド: (a) 2 つの実数値 x, y から上のような複素数を作るメソッド。(ヒント: 新しく配列を作り、それを返す。)
   (b) 2 つの複素数を足すメソッド。
   (c) 2 つの複素数を掛けるメソッド。(ただし、複素数 a の実部、虚部を $a_{\mathrm{re}}, a_{\mathrm{im}}$ としたとき $ab = a_{\mathrm{re}}b_{\mathrm{re}} - a_{\mathrm{im}}b_{\mathrm{im}} + i(a_{\mathrm{re}}b_{\mathrm{im}} + a_{\mathrm{im}}b_{\mathrm{re}})$ である。)
   (d) 複素数 a の絶対値 $|a| \equiv \sqrt{{a_{\mathrm{re}}}^2 + {a_{\mathrm{im}}}^2}$ を求めるメソッド。
   (e) 複素数 $x + iy$ を "x+y`i`" のように表示するメソッド。

   計算: 自分の学生証番号を 8 桁の整数 $uvwxyz$ だとする。3 つの複素数が $a = u + iv, b = w + ix, c = y + iz$ であるとき、以下の値を計算し、表示させよ。

   (ア) $b + c$　(イ) ab　(ウ) ac　(エ) $ab + ac$　(オ) $a(b + c)$

2. (optional)

   複素数 z, c が与えられたとき、$z \leftarrow z^2 + c$ という操作を行って z を変化させていくとする。

   - 複素数の関数 $f(z, c) = z^2 + c$ を定義せよ。

- 上の操作に従って z を変化させていって z の絶対値が 2 以上になるのに要した操作の回数を求める関数 $n(z, c)$ を定義せよ。ただし、30 回以上操作しても 2 以上にならない場合は「-1 回」だとする。
- 次の値を求めよ: $n(0, -1.7 + 0.1i), \quad n(0, -1.3 + 0.1i), \quad n(0, -1 + 0.1i)$

3. (すごく optional)

   複素平面上の点 p に対して、$n(0, p) = -1$ となるような p の集合をマンデルブロ (Mandelbrot) 集合という。$p = a + ib$ $(-2 \leq a \leq 0.5,\ -1.25 \leq b \leq 1.25)$ の領域のマンデルブロ集合を図示するプログラムを作れ。ただし、$n(0, p) \geq 0$ なる点 p は $n(0, p)$ の値に応じた色で塗り、$n(0, p) = -1$ なる点 p は黒で塗ることにする。

   例えば 200×200 の点から成る画面を複素平面に対応させるには、画面上の座標 (x, y) から

   $$p_{xy} = \frac{(a_{\max} - a_{\min})}{w} x + a_{\min} + i \left(\frac{(b_{\max} - b_{\min})}{h} y + b_{\min} \right)$$

   (ただし $a_{\min}, a_{\max}, b_{\min}, b_{\max}$ は実部・虚部の最小最大値、w, h は画面の幅と高さ) のようにして決めてやればよい。また、図示の方法は web ページに掲載したプログラムを参考にせよ。

4. (さらに optional)

   関数 f の定義を変えるとマンデルブロ集合の形も変わる。例えば、$f(z, c) = z^k + c$ としたときの k を変えるとマンデルブロ集合の形はどのように変わるかを試せ。