K会2012年度夏期講習　情報講座

中級編Part2　再帰呼び出し

命令の中で自分自身を呼び出す「再帰呼び出し」を学ぶ。

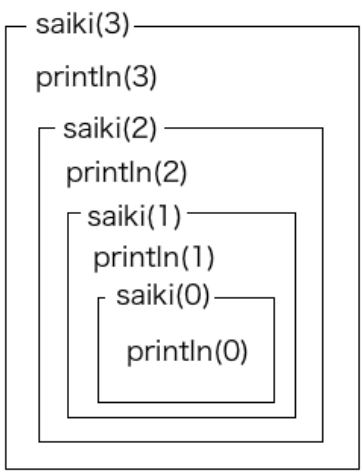
目次

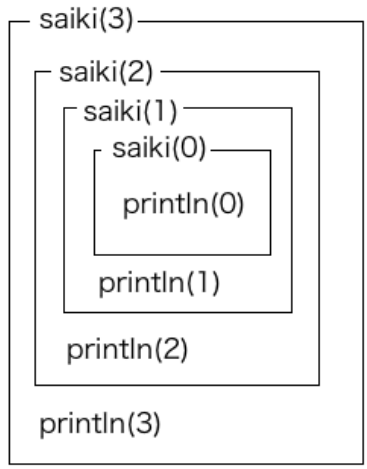
2-1. 再帰呼び出しの基本

再帰呼び出しは、命令のなかでその命令自身を呼び出すテクニックのことである。下のプログラムを書こう。

```
void saiki(int n){
  println(n);
  if(n>0){
    saiki(n-1);
  }
}

void setup(){
  noLoop();
}

void draw(){
  saiki(10);
}
```

この命令 saiki は、引数としてとった n を出力して、もし n が 0 より大きければ、n より 1 つ小さい数を引数としてさらに saiki を呼び出す。上の例では最初の引数は 10 なので、n が 10, n が 9, … n が 0 の順に 11 段階呼び出されてそこで止まる。(下図の左は saiki(3)を呼び出した時の様子)





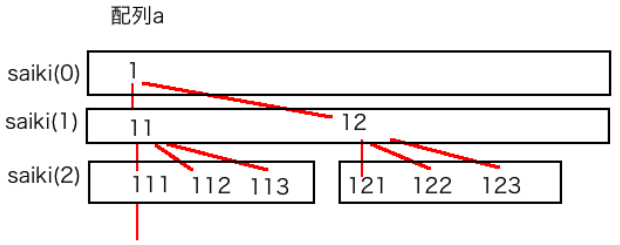
ここで、saiki 命令の中の、println(n)と if ブロックとの順番を入れ替えると出力はどう変わるか試してみよ。このとき saiki(n-1)を呼び出して、その呼び出しが終わった後で n を出力することになるので、上図の右のようになるはずである。

（１）次のような再帰関数 void saiki(int n)を書け。
　引数として整数 n をとり、n を出力し、n が 1 より大きいとき、n が 2 で割り切れたら saiki(n/2)、割り切れなければ saiki(n+1)で再帰呼び出しする。

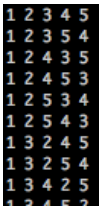
例：saiki(9)　→　9, 10, 5, 6, 3, 4, 2, 1
　　saiki(13) →　13, 14, 7, 8, 4, 2, 1

（２）次のプログラムを書こう。
　これは 1 から 5 までの数を使ってできる 5 桁の数を小さい順に全て表示するものである。どうしてそうなるのか、命令 saiki が何をしているのかを納得せよ。

```
int[] a;

void saiki(int n){
  if(n==5){
    println(a[0]+" "+a[1]+" "+a[2]+" "+a[3]+" "+a[4]);
  }else{
    for(int i=1; i<=5; i++){
      a[n] = i;
      saiki(n+1);
    }
  }
}

void setup(){
  a = new int[5];
  noLoop();
}

void draw(){
  saiki(0);
}
```



（３）前のプログラムを書き換えて、1, 2, 3, 4, 5 を並び替えてできる 5 桁の数を小さい順に全て列挙せよ。（前のプログラムで、同じ数字を２個以上含ませないように改良する）



（４）1, 2, 3, 4, 5 を並び替えてできる 5 桁の整数のなかで、7 で割り切れるものを小さい順に全て列挙せよ（全部で 15 個ある）。

2-2. 数学の問題を解く

再帰呼び出しを使って、数学の問題を計算してみよう。ものによっては普通に for 文を書いたほうが速いものもあるが、練習と思ってやってみよう。

（１）階乗の計算

階乗とは 1 からその数までの積のことで、5 の階乗なら 1*2*3*4*5 = 120 である。

右のプログラムの kaijo 命令は階乗を計算する。

kaijo(0)　は　1
kaijo(n)　は　n * kaijo(n-1)　ただし n>0

```
int kaijo(int n){
  if(n==0){
    return 1;
  }else{
    return n*kaijo(n-1);
  }
}

void setup(){
  noLoop();
}

void draw(){
  println(kaijo(5));
}
```

n の階乗なら、n に n-1 の階乗をかけたもの、というわけだ。難しいことはない。

このように、hoge(n)が hoge(n-1)などを使って書かれるときに、再帰呼び出しは役に立つ。

（２）フィボナッチ数列

次の再帰関数 int f(int n)を作れ。

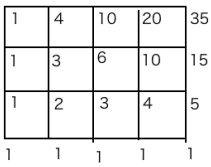
f(0)　は　1
f(1)　は　1
f(n)　は　f(n-1) + f(n-2)　　ただし n>1

```
f(0) = 1
f(1) = 1
f(2) = 2
f(3) = 3
f(4) = 5
f(5) = 8
f(6) = 13
f(7) = 21
f(8) = 34
f(9) = 55
```

これによってできる 1, 1, 2, 3, 5 … という数列をフィボナッチ数列という。K 会の数学コースをとってる人なら知ってるかも。

（３）場合の数

n*m のマス目状の道を、左下から右上まで行く場合の数を計算する再帰命令 int michi(int n, int m)を作れ。



michi(n, 0)　は　1
michi(0, m)　は　1

michi(n, m)　は　michi(n-1, m) + michi(n, m-1)　　ただし 0 < m , n

例：michi(3, 2) = 10,　michi(4, 3) = 35,　michi(10, 8) = 43758

（４）n 個のものから m 個を選び出す場合の数を計算する再帰命令を書け。
　ヒント：○|○○●○●○○○○●　一番左の 1 個を使わない
　　●|○○○○○○●●○○　使う

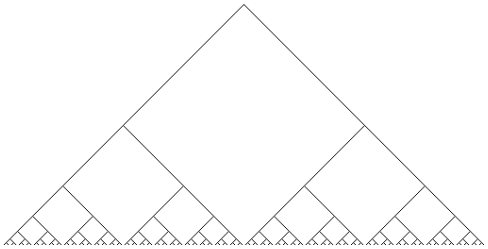
（５）1 以上 m 以下の数を足しあわせて、合計 n になる場合の数を答える再帰命令 int saiki(int n, int m)を書け。

　例：n=5, m=3 のとき　…　5 通り
　　1 1 1 1 1
　　2 1 1 1
　　2 2 1
　　3 1 1
　　3 2
　例：n=4, m=4 のとき　…　5 通り
　　1 1 1 1
　　2 1 1
　　2 2
　　3 1
　　4

2-3. フラクタル図形

大きく見ても細かく見ても同じようなパターンが現れている図形をフラクタル図形という。このような図形は再帰を使って描くことができる。

```
void f(int w, int x, int y, int n){
  if(n<10){
    line(x, y, x-w, y+w);
    line(x, y, x+w, y+w);
    f(w/2, x-w, y+w, n+1);
    f(w/2, x+w, y+w, n+1);
  }
}
```



この再帰命令は、右のパターンを描く命令である。
引数で与えられた点(x, y)から、

(x-w, y+w)

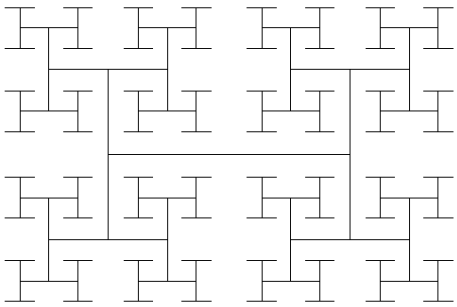
(x+w, y+w)

の二点に向けて直線を引き、
その終点で再帰呼び出しを行う。

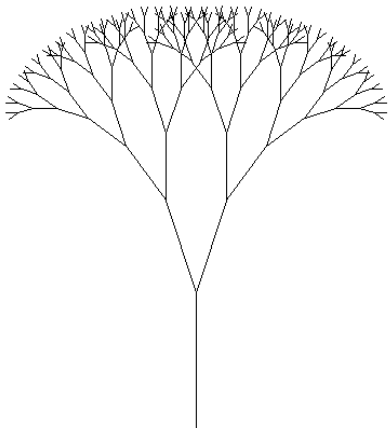
再帰が深くなるごとに w を小さく（ここでは半分）にしていくことで、どんどん細かくなっていくパターンになる。変数 n は再帰の深さを表していて、上のプログラムでは 10 段階以上は再帰しないようにしてある。

下のパターンがどのように書かれているかを考えて、これらを表示するプログラムを書いてみよう。

（１）H 字再帰



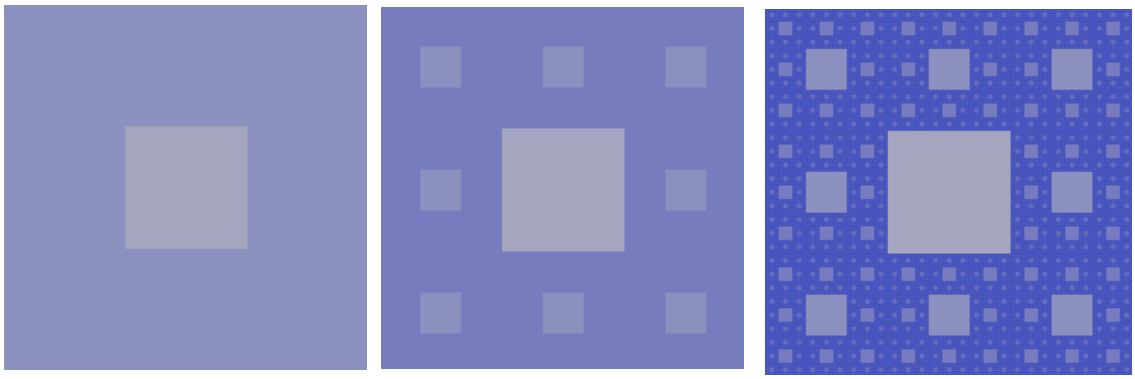
（２）木



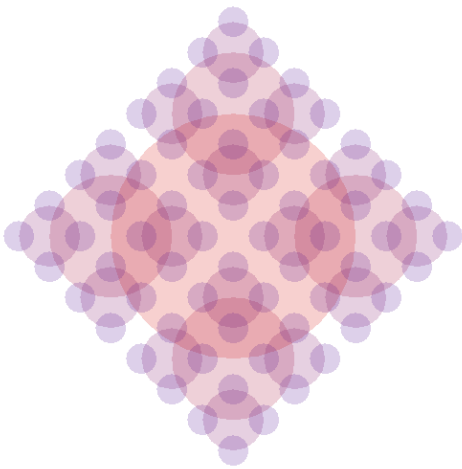
（３）シェルピンスキーのカーペット
半透明の四角形を並べる。
再帰命令 void f(x, y, w, n)は、左上を(x, y)とする一辺 w の正方形を描いて、

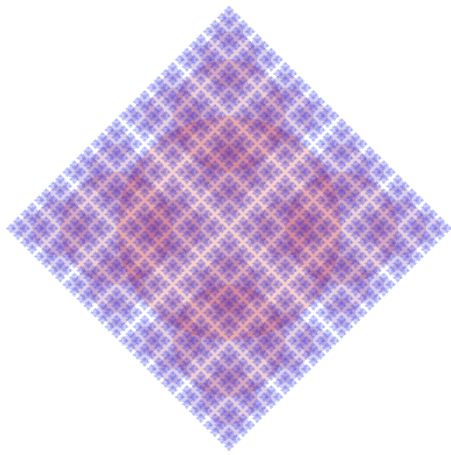
■■■
■□■
■■■

の■の位置に対して再帰呼びだしを行う。これを繰り返すと下の一番右のようなパターンになる。



（４）テーブルクロス
再帰が深くなるごとに色も変えてみる。





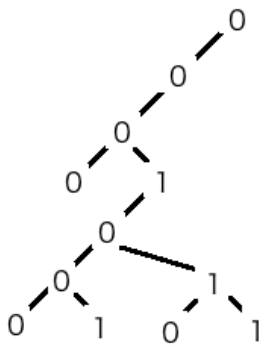
2-4. 深さ優先探索

（講義）ネットワーク、木構造、深さ優先探索

（１）数学パズル
0 と 1 を 10 文字並べて、どの連続する３つをとっても同じ並びが現れないようにしたい。
　　例：01010 は最初の３文字と最後の３文字がどちらも 010 なのでダメ
このような 10 文字の 01 列のなかで、その列を数字としてみた時に一番小さいものを求めよ。
　　例：01101 は 00110 よりも大きい

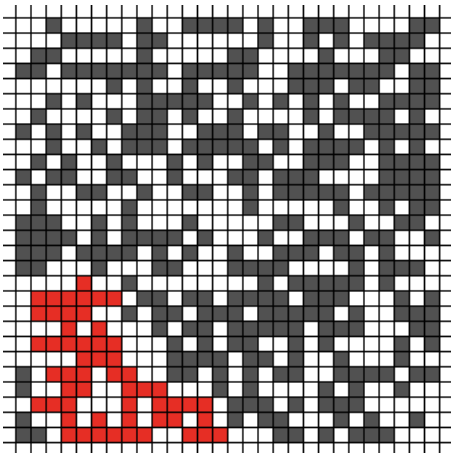
・再帰命令の方針
再帰命令 int saiki(int n)は、数列の n 文字目を決定する。もし成功した例を見つけたら、1 を返して、失敗だったら 0 を返す。今調べてる数列をグローバルな配列 a[]に記録しておくとして、
(i) n が 10 なら 1 を返す。
(ii) a[n]を 0 にしてみる。
(iii) もし a[n-2], a[n-1], a[n]の３文字の並びがもっと前に現れていなければ、saiki(n+1)を呼び出す。この結果が 1 だったら 1 を返して終了。
(iv) a[n]を 1 にしてみる。
(v) もし a[n-2], a[n-1], a[n]の３文字の並びがもっと前に現れていなければ、saiki(n+1)を呼び出す。この結果が 1 だったら 1 を返して終了。
(vi) 0 を返す。

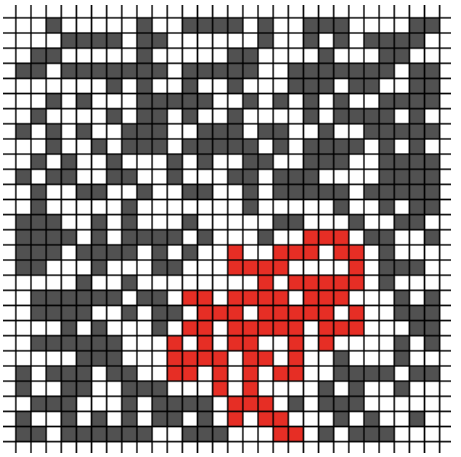
　下の階層から 1 が帰ってきた時点でそのルートが正解だったことがわかるので、1 をすぐに返して再帰を脱出する。正解が判明した後も探索を続けると、グローバル配列 a に記録していた正解が壊れてしまうので、探索をその時点で打ち切らないといけない。
　再帰を脱出したら、配列 a を出力すれば良い。



（２）連鎖判定

あるマスにつながっているマスたちを探索するプログラムを書こう。
クリックした黒マスに連結しているマスたちを赤く塗りつぶしたい（斜めはつながっているとはみなさないことにする）





まず、二次元配列 a にランダムに整数 0, 1 を記録したものを作ろう。これが 0 なら白、1 なら黒としよう。
クリックしたのがどのマスかは、マウスの位置をマスの横幅で割ればわかる。

そして、連結しているマスを探す再帰命令を作る。
再帰命令を呼び出す前に、もう一つの二次元配列 b を用意して、これは全部 0 としておく。ここには「すでに調べたマス」を 1 として記録することにする。
再帰命令 void tansaku(int x, int y)は、

(i) b[x][y]が 1 だったらそこで終了（return;）
(ii) a[x][y]が 0 だったら白マスなのでそこで終了（return;）
(iii) x, y が領域外に飛び出していたら終了（return;）
(iv) b[x][y]を 1 にする（そこをすでに見たことにする）
(v) (x, y)の上下左右のマスに対して tansaku を再帰呼び出しする

とすればよい。

探索が終了したら、b[i][j]が 1 となってるマスを赤で塗りつぶせば OK だ。

このような探索は、パズルゲームによく使われる（いくつつながったら消える、とか）

（3）8 クイーン問題

チェスのコマのクイーンは、縦横斜めにいるコマを取ることができる（将棋の飛車と角を合わせたような）。8*8 の盤面に、どのクイーンも他のクイーンを取れないように 8 個のクイーンを配置する問題を 8 クイーン問題という。この問題を再帰を使って解け。

グローバル変数（二次元配列）として盤面の状態を持つ。
再帰命令では、置くことができる場所にクイーンをおいて、次の再帰を呼ぶ。その再帰が終わって（正解じゃなければ）置いたクイーンを除去して、次の置ける場所において再帰呼び出しを行う。これを繰り返す。すべてのクイーンが置ききれたら、正解だという信号を return すればいい。

盤面の大きさが 8*8 と 10*10 のときの正解例。
普通にプログラムを書くと、10*10 はやや時間がかかる。それ以上の大きさは終わらないかも。