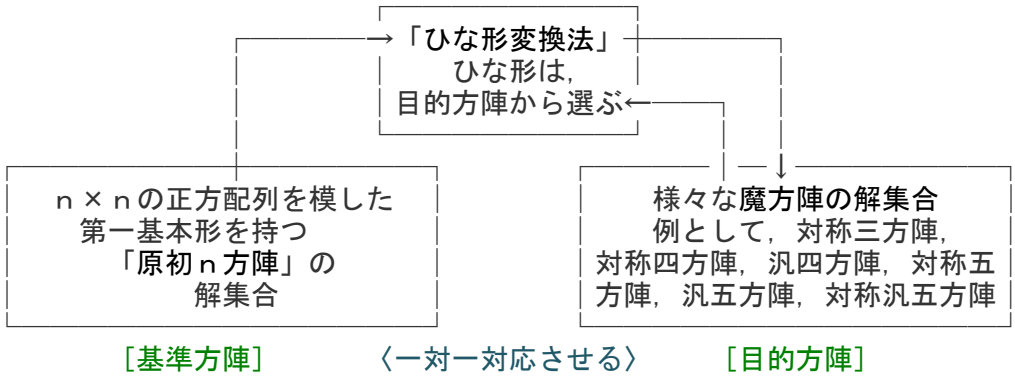
第４部：　『魔方陣の最新研究』：摂田 寛二
第１章：　原初方陣とひな形変換法の基礎的研究

今回の研究では，新しい概念と方法論を導入する。原初方陣とひな形変換法である。
「原初方陣」とは，その第一基本形が規則的に順序付けられたｎ×ｎの正方配列を模した方陣だ。最初から行列の定和を放棄しているので，それ自身は決して「魔方陣」ではないが，基準方陣として様々なタイプの魔方陣に対応する特長を有する。
そしてその対応付けを可能にするのが，方陣研究の新しい技法である「ひな形変換法」である。



第１節：原初三方陣とひな形変換法

＃１．最初に三方陣の場合を調べる。三方陣の解は，対称型の１個しか無いから，「一対一対応」を言っても無意味かもしれないが，三方陣を例外にするわけにはいかない。最小方陣でも成り立つことを確めておかなければならない。それが理論的研究てぇものだからだ。
そのために下表のような「原始解」８個を用意する。反転・回転形を整理する前の形だ。

** 目的方陣：補数対称型の平面三方陣（原始解８個）**

0/	1/	2/	3/	4/	5/	6/	7/	8/
1 2 3	4 9 2	2 9 4	6 7 2	2 7 6	8 3 4	4 3 8	8 1 6	6 1 8
4 5 6	3 5 7	7 5 3	1 5 9	9 5 1	1 5 9	9 5 1	3 5 7	7 5 3
7 8 9	8 1 6	6 1 8	8 3 4	4 3 8	6 7 2	2 7 6	4 9 2	2 9 4

上図のような目的方陣８個に対応する原初三方陣を８個最初に作り，それを逐一「ひな形変換」したい。そうして目的方陣８個を再構成することができるかどうかを調べたい。
では，その「原初方陣」とはどんなものか。第一基本形が「配列」と同じ形をしているということだけはわかっている。ただし，その第一基本形は，「基本図」として変数の名前を指すだけでなく，具体的な方陣として名前と同じ数値を取る場合もあるということだ。

試みにこの第一基本形を反転したり回転したりして，８個の異形を作ってみよう。

1/	2/	3/	4/	5/	6/	7/	8/
1 2 3	7 4 1	9 8 7	3 6 9	1 4 7	3 2 1	9 6 3	7 8 9
4 5 6	8 5 2	6 5 4	2 5 8	2 5 8	6 5 4	8 5 2	4 5 6
7 8 9	9 6 3	3 2 1	1 4 7	3 6 9	9 8 7	7 4 1	1 2 3

どうやらこれが「原初三方陣」らしい。この８形の諸性質を先ず調べてみよう。
一見してわかるように行列の定和が成り立っていないから，原初方陣は「魔方陣」ではな

い。しかし，既に補数対称型であり，汎対角線の定和さえもが成り立っている。様々な面白い性質も見つかる。

1．先ず各行列の和を調べよう。
　１＋２＋３＝６，　４＋５＋６＝15，　７＋８＋９＝24，
　１＋４＋７＝12，　２＋５＋８＝15，　３＋６＋９＝18
というふうに，行列の定和は成り立っていない。
　わずかに和が 15 になる中央行列の２本だけが定和を成り立たせている）。

2．しかし，方陣の幾何学的中心を中心点として対称の位置にある２数ずつ４組の和は，
　１＋９＝10，　２＋８＝10，　３＋７＝10，　４＋６＝10
と，一定の値を示し，「10 の補数」を成している。
　つまり，この原初三方陣は「補数対称型三方陣」である。
　また，方陣空間を拡張して見るとわかるが，
　１＋５＋９＝15，　１＋６＋８＝15，　２＋４＋９＝15，
　２＋６＋７＝15，　３＋４＋８＝15，　３＋５＋７＝15
というふうに，汎対角線の定和が既に成り立っている。
　つまり，この原初三方陣は「汎三方陣」である。

　配列に連続する自然数をただ順に並べただけなのに，原初方陣は何の苦労も無く，連立型の「補数対称型かつ汎対角線型の三方陣」（短くして言うと「対称汎三方陣」）を実現してしまっている！
　これは，考えてみれば，不思議な話である。魔方陣の作成に平素苦労している人ほど，そう言われてあらためて驚くに違いない。
　ただ，行列の定和が成り立っていないので，原初方陣を「魔方陣」とは言わない。我々は伝統的に行列の定和を最優先して来た。他のどの条件を犠牲にしても，行列の定和を先ず実現して来た。３次の魔方陣では，汎対角線の定和をあえて棄てて，行列の定和と対称性を残したのである。
　その伝統が長く続いたので，我々は原初方陣のこの希有の性質をつい過小評価し続けて来た。そして，結果的に無視・差別して来た。我々がふだん何気なく吸っている空気を意識しないように，その特性すら無自覚に忘れていたように思う。

3．原初三方陣には，他にも面白い性質がいろいろある。
　例えば，「クロスサムの等和」。最初から式で示そう。
　１＋５＝２＋４，　２＋６＝３＋５，　４＋８＝５＋７，　５＋９＝６＋８
任意の２×２の内包二方陣の対角２数の和がお互いに等しいのである。定和ではないが，等和なのである。
　１＋９＝３＋７　が成り立つから，内包三方陣のクロスサムも等しい。

　また同じ行内，列内では，外側の２数の和が，中央数の２倍に等しい。
　１＋３＝２×２，　４＋６＝５×２，　７＋９＝８×２，
　１＋７＝４×２，　２＋８＝５×２，　３＋９＝６×２，．．．．．
　そのほかには，　１＋３＋７＋９＝２＋４＋６＋８＝20　なども成り立つ。

4．我々が一般に「三方陣」と言う時には，行列の定和（＝15）も成り立つような「魔方陣」を意味する。魔方陣は，特別の存在である（だから，何千年もの間，知的な人々の特別の興味を引き付け続けて来た）。
　では，原初方陣とどこが同じで，どこが違うのか。
（１）１～９の連続する自然数を重複無く全個用いる。これは，同じだ。
（２）原初方陣第一基本形の並び方は原則として１つだが，魔方陣には様々なパターンがあり，並べ替えもできる。
（３）魔方陣では行列の定和が成り立つ。ここが一番違う。

（４）補数対称型方陣である。これは，共通だ。
（５）汎対角線上の３数の定和が常に成り立つのは，原初方陣。
　魔方陣は，主対角線の２本のみが定和を成り立たせる。

５．試みに正方配列の中で原初三方陣の数字を動かし，並び替えができるようにしてみよう。
　無条件に並べ替えると，９×８×７×‥‥×２×１通りの方陣が出来てしまう。
（１）補数の対称配置を残そう。つまり，
　n１＋n９＝n２＋n８＝n３＋n７＝n４＋n６＝10　を守るものとすると，
「10 の補数」ペア４組がいつも一緒に動くので，
　　2^4 × ４！＝ 16 × 24 ＝ 384 通りの並べ替えができる。
（２）加えて，汎対角線３数の定和が成り立つようにしてみよう。
　　n１＋n５＋n９＝15,　n１＋n６＋n８＝15,　n２＋n４＋n９＝15,
　　n２＋n６＋n７＝15,　n３＋n５＋n７＝15,　n３＋n４＋n８＝15
が成り立つとすると，次の８通りの並び方しか無くなる。

1/	2/	3/	4/	5/	6/	7/	8/
1 2 3	1 4 7	3 2 1	3 6 9	7 4 1	7 8 9	9 6 3	9 8 7
4 5 6	2 5 8	6 5 4	2 5 8	8 5 2	4 5 6	8 5 2	6 5 4
7 8 9	3 6 9	9 8 7	1 4 7	9 6 3	1 2 3	7 4 1	3 2 1

（３）この対称汎三方陣に，行列の定和まで持たせることはできない。行列の定和を実現するとなれば，汎対角線の定和をあきらめなければならない。
（４）原初方陣に戻って，クロスサムの等和を成り立たせたまま並び替えができるか。
　いま現節（２）の汎対角線の定和式を組み合わせると，

n１＋n５＋n９＝15	n２＋n６＋n７＝15	n３＋n５＋n７＝15
−) n２＋n４＋n９＝15	−) n３＋n５＋n７＝15	−) n３＋n４＋n８＝15
n１＋n５＝n２＋n４,	n２＋n６＝n３＋n５,	n４＋n８＝n５＋n７

　n１＋n５＋n９＝15
−) n１＋n６＋n８＝15
　n５＋n９＝n６＋n８

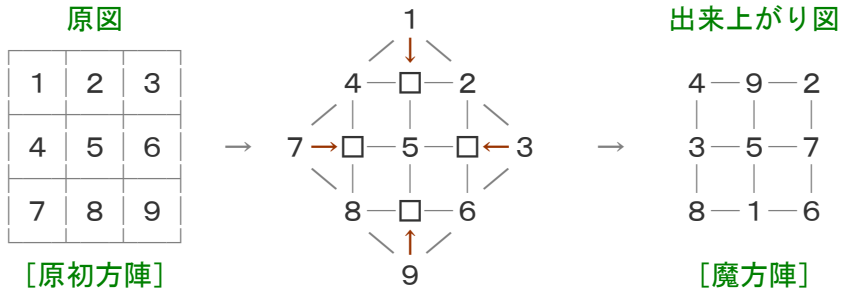
逆もまた可なり。
n１＋n５＝n２＋n４　の両辺に n９ をたすと，
　n１＋n５＋n９＝n２＋n４＋n９
n２＋n６＝n３＋n５　の両辺に n７ をたすと，
　n２＋n６＋n７＝n３＋n５＋n７　が言える。
‥‥‥‥

　つまり，クロスサム等和と汎対角線定和は，つねに一緒に成り立つことがわかる。だから，クロスサム等和を守ったまま並び替えると，（２）と同じ８通りの異形が出来るのである。

６．ところで，３次の魔方陣には７つの異形があり，単に回転させたり反転させただけのものだが，全部で８通りの形があることがわかっている。原初方陣の８形と何か関連があるのだろうか。

　ここで思い出すのが，三方陣の古典的構成法である。下図を見てほしい。
（１）原図を 45 度ほど時計回りに回転させ，中央図のひし形方陣を得る。
（２）中央部の５数を切り取り，□のような間隔を取って方陣に並べる。その□の場所にははみ出したひし形頂点の４数を反対側の遠い方の□に埋め込む。
（３）出来上がり図は，三次の魔方陣である。行列の定和と主対角線２本の定和が成り立っていることを確かめてほしい。対称性も保たれている。
（４）汎対角線の定和はもはや成り立たない。原初方陣の汎対角線の定和式が，魔方陣の行列の定和式に転用されたからである。

［三方陣の古典的構成法］



この構成法の意義は，原初方陣を魔方陣に並べ替える「**変換法**」だと言うことだ。
では，原初方陣の最初のリストの８形各個に，この変換法を施してみよう。

```
[1]                1                #
  1 2 3          4 # 2            4 9 2          4 9 2
  4 5 6 --> 7 # 5 # 3 --> # 3 5 7 # --> 3 5 7
  7 8 9          8 # 6            8 1 6          8 1 6
                   9                #

[2]                7                #
  7 4 1          8 # 4            8 3 4          8 3 4
  8 5 2 --> 9 # 5 # 1 --> # 1 5 9 # --> 1 5 9
  9 6 3          6 # 2            6 7 2          6 7 2
                   3                #

[3]                9                #
  9 8 7          6 # 8            6 1 8          6 1 8
  6 5 4 --> 3 # 5 # 7 --> # 7 5 3 # --> 7 5 3
  3 2 1          2 # 4            2 9 4          2 9 4
                   1                #

[4]                3                #
  3 6 9          2 # 6            2 7 6          2 7 6
  2 5 8 --> 1 # 5 # 9 --> # 9 5 1 # --> 9 5 1
  1 4 7          4 # 8            4 3 8          4 3 8
                   7                #

[5]                1                #
  1 4 7          2 # 4            2 9 4          2 9 4
  2 5 8 --> 3 # 5 # 7 --> # 7 5 3 # --> 7 5 3
  3 6 9          6 # 8            6 1 8          6 1 8
                   9                #

[6]                3                #
  3 2 1          6 # 2            6 7 2          6 7 2
  6 5 4 --> 9 # 5 # 1 --> # 1 5 9 # --> 1 5 9
  9 8 7          8 # 4            8 3 4          8 3 4
                   7                #

[7]                9                #
  9 6 3          8 # 6            8 1 6          8 1 6
  8 5 2 --> 7 # 5 # 3 --> # 3 5 7 # --> 3 5 7
  7 4 1          4 # 2            4 9 2          4 9 2
                   1                #
```

```
[8]              7              #
  7 8 9        4 # 8          4 3 8          4 3 8
  4 5 6 --> 1 # 5 # 9 --> # 9 5 1 # --> 9 5 1
  1 2 3        2 # 6          2 7 6          2 7 6
                 3              #
```

出来上がった８個の解は，まさに対称三方陣の解集合そのものである。
かくして，２つの異種方陣：原初三方陣と対称三方陣が同じ１つの変換法によって結び付けられた。しかも，８つの解の間には美しい「一対一対応」の関係が見られるのである。
そして，くしくも古典的構成法が目指したのは，対称三方陣の「ひな形」であった。

このような一対一対応の関係が可能なのは，２つの異種方陣に構造的な共通性があるためと思われる。
一方の原初方陣は，行列の定和をあきらめた対称汎方陣。他方の魔方陣は，汎対角線定和を手放す代わりに行列定和を手に入れた対称方陣なのだ。あるいは，原初方陣の性質を分有する同系方陣である，と言っても良いかもしれない。

＃２．ところで，変換法もひな形も，決して１つに限られるものではない。上の「古典的構成法」を必ずしも使わなくてもよいのである。別の方法を考えてみよう。
試みに対称三方陣のリストの第１解を「ひな形」と決めて，あらためてひな形変換法と原初方陣の構成を考えてみることにしよう。

```
/原初方陣  /ひな形      /* ひな形変換法 */
 ._._._.   ._._._.
 |1|2|3|   |4|9|2|     void damt1(){
 |-+-+-|   |-+-+-|       n[1]=m[4]; n[2]=m[9]; n[3]=m[2];
 |4|5|6|   |3|5|7|       n[4]=m[3]; n[5]=m[5]; n[6]=m[7];
 |-+-+-|   |-+-+-|       n[7]=m[8]; n[8]=m[1]; n[9]=m[6];
 |7|8|9|   |8|1|6|     }
 '-'-'-'   '-'-'-'
```

左上の原初方陣から右上のひな形の解を得る変換法が「ひな形変換法」である。このように一番簡単でエレガントな方法を，我々は提案した。
では，ひな形から原初方陣を，１個ではなく全数個，計算で求める別の変換法は無いか。

```
/原図                    /ひな形
 ._._._.   n1+n2+n3=15;   ._._._.   n4+n9+n2=15;
 |1|2|3|   n4+n5+n6=15;   |4|9|2|   n3+n5+n7=15;
 |-+-+-|   n7+n8+n9=15;   |-+-+-|   n8+n1+n6=15;
 |4|5|6|   n1+n4+n7=15;   |3|5|7|   n4+n3+n8=15;
 |-+-+-|   n2+n5+n8=15;   |-+-+-|   n9+n5+n1=15;
 |7|8|9|   n3+n6+n9=15;   |8|1|6|   n2+n7+n6=15;
 '-'-'-'   n1+n5+n9=15;   '-'-'-'   n4+n5+n6=15;
      and  n3+n5+n7=15;        and  n2+n5+n8=15;
```

思い出してほしい。標準型三方陣の解を求める一番普通の方法は，上の原図に付せられた連立方程式を計算で解くのであった。それによって，右側のような解が得られた。
この方法を「関数を解く手順」と考えれば，右上図の解に付せられた連立方程式を計算で解いて左上図のような「原図」を求める方法は，「逆関数を解く手順」と考えられる。
そこで，次のようなプログラムを組んで，この逆関数をコンピューターに実際に解かせてみた（正関数を解くプログラムと逆関数を解く方法を比較のために両方載せておいた）。

```
/* 標準型三方陣を構成するプログラム */
/** Dictated in 2012 **/
/* Set n2 */
```

```
void stp01(){
 short a;
  for(a=9;a>0;a--){
```

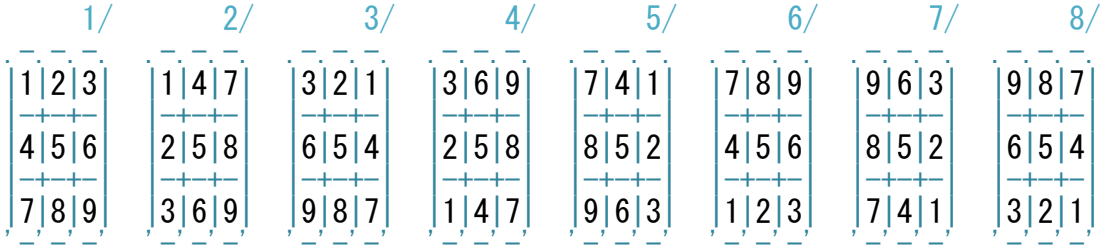
```
    if(uflg[a]==0){
      nm[2]=a; uflg[a]=1;
      stp02();
      uflg[a]=0;}
  }
}
/**/
/* Set n5 */
void stp02(){
 short a;
  for(a=1;a<10;a++){
    if(uflg[a]==0){
      nm[5]=a; uflg[a]=1;
      stp03();
      uflg[a]=0;}
  }
}
/**/
/* Set n8=15-n2-n5 */
void stp03(){
 short a;
  a=15-nm[2]-nm[5];
  if((0<a)&&(a<10)){
    if(uflg[a]==0){
      nm[8]=a; uflg[a]=1;
      stp04();
      uflg[a]=0;}
  }
}
/**/
/* Set n4 */
void stp04(){
 short a;
  for(a=1;a<10;a++){
    if(uflg[a]==0){
      nm[4]=a; uflg[a]=1;
      stp05();
      uflg[a]=0;}
  }
}
/**/
/* Set n6=15-n4-n5 */
void stp05(){
 short a;
  a=15-nm[4]-nm[5];
  if((0<a)&&(a<10)){
    if(uflg[a]==0){
      nm[6]=a; uflg[a]=1;
      stp06();
      uflg[a]=0;}
  }
}
/**/
/* Set n1 */
void stp06(){
 short a;
  for(a=1;a<10;a++){
    if(uflg[a]==0){
      nm[1]=a; uflg[a]=1;
      stp07();
      uflg[a]=0;}
  }
}
/**/
/* Set n3=15-n1-n2 */
void stp07(){
 short a;
  a=15-nm[1]-nm[2];
  if((0<a)&&(a<10)){
    if(uflg[a]==0){
      nm[3]=a; uflg[a]=1;
      stp08();
      uflg[a]=0;}
  }
}
/**/
/* Set n7=15-n1-n4 */
void stp08(){
 short a,b;
  a=15-nm[1]-nm[4];
  b=15-nm[3]-nm[5];
  if((a==b)&&(0<a)&&(a<10)){
    if(uflg[a]==0){
      nm[7]=a; uflg[a]=1;
      stp09();
      uflg[a]=0;}
  }
}
/**/
/* Set n9=15-n7-n8 */
void stp09(){
 short a,b,c;
  a=15-nm[7]-nm[8];
  b=15-nm[3]-nm[6];
  c=15-nm[1]-nm[5];
  if((a==b)&&(a==c)){
    if((0<a)&&(a<10)){
      if(uflg[a]==0){
        nm[9]=a; uflg[a]=1;
        solrecord();
        uflg[a]=0;}
    }
  }
}
/**/
/* Record The Solutions */
void solrecord(){
 short n;
  tcnt++;
  sn[tcnt][0]=tcnt;
  for(n=1;n<10;n++){sn[tcnt][n]=nm[n];}
}
/**/

/* 原初三方陣を構成するプログラム */
/** Dictated in 2012 **/
/* Set n1 */
void pstp01(){
```

```
 short a;
  for(a=1;a<10;a++){
    if(uflg[a]==0){
      nm[1]=a; uflg[a]=1;
      pstp02();
      uflg[a]=0;}
  }
}
/**/
/* Set n5 */
void pstp02(){
 short a;
  for(a=1;a<10;a++){
    if(uflg[a]==0){
      nm[5]=a; uflg[a]=1;
      pstp03();
      uflg[a]=0;}
  }
}
/**/
/* Set n9=15-n1-n5 */
void pstp03(){
 short a;
  a=15-nm[1]-nm[5];
  if((0<a)&&(a<10)){
    if(uflg[a]==0){
      nm[9]=a; uflg[a]=1;
      pstp04();
      uflg[a]=0;}
  }
}
/**/
/* Set n2 */
void pstp04(){
 short a;
  for(a=1;a<10;a++){
    if(uflg[a]==0){
      nm[2]=a; uflg[a]=1;
      pstp05();
      uflg[a]=0;}
  }
}
/**/
/* Set n8=15-n2-n5 */
void pstp05(){
 short a;
  a=15-nm[2]-nm[5];
  if((0<a)&&(a<10)){
    if(uflg[a]==0){
      nm[8]=a; uflg[a]=1;
      pstp06();
      uflg[a]=0;}
  }
}
/**/
/* Set n4=15-n2-n9 */
void pstp06(){
 short a;
  a=15-nm[2]-nm[9];
  if((0<a)&&(a<10)){
    if(uflg[a]==0){
      nm[4]=a; uflg[a]=1;
      pstp07();
      uflg[a]=0;}
  }
}
/**/
/* Set n6=15-n4-n5 */
void pstp07(){
 short a,b;
  a=15-nm[4]-nm[5];
  b=15-nm[8]-nm[1];
  if((a==b)&&(0<a)&&(a<10)){
    if(uflg[a]==0){
      nm[6]=a; uflg[a]=1;
      pstp08();
      uflg[a]=0;}
  }
}
/**/
/* Set n3=15-n8-n4 */
void pstp08(){
 short a;
  a=15-nm[4]-nm[8];
  if((0<a)&&(a<10)){
    if(uflg[a]==0){
      nm[3]=a; uflg[a]=1;
      pstp09();
      uflg[a]=0;}
  }
}
/**/
/* Set n7=15-n3-n5 */
void pstp09(){
 short a,b;
  a=15-nm[3]-nm[5];
  b=15-nm[2]-nm[6];
  if((a==b)&&(0<a)&&(a<10)){
    if(uflg[a]==0){
      nm[7]=a; uflg[a]=1;
      ansrecord();
      uflg[a]=0;}
  }
}
/**/
/* Record The Answers */
void ansrecord(){
 short n;
  ps[cnt][0]=cnt+1;
  for(n=1;n<10;n++){ps[cnt][n]=nm[n];}
  cnt++;
}
/**/
```

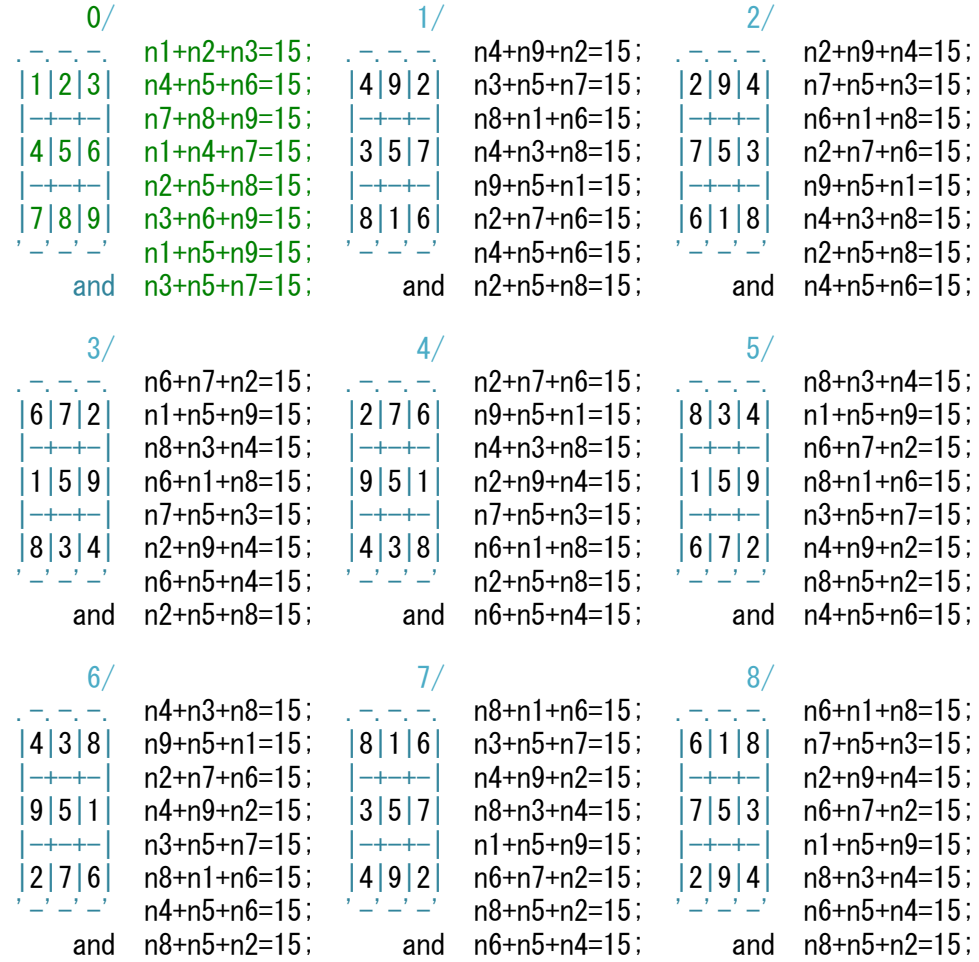
得た結果は，以下の通り。他の方法で得た原初三方陣と同じ解集合が得られた。

** 上のプログラムで構成した原初三方陣の解８個 **



次の疑問は，果たして他のひな形でも同じ結果が得られるかということだが，その心配は要らない。次の表を見てほしい。各ひな形について定義式を一々書き出したものだ。

** 各ひな形から得られる原初三方陣用の定義式群 **



式の出る順番が違うだけで，内容が全く同じことにお気づきだろうか。
これならば，上と同じ原初方陣の解集合を出力するしかないではないか。
つまり，原初三方陣８個の解集合は，一通りしか無いのである。

あらためて「原初方陣とは一体何なのか」と，思わされる。

ここで「基本図の変形」について触れておこう。
下の表は，（１）90 度ずつの回転と（２）主対角線の１本を軸にした反転という２つの操作を組み合わせて，「基本図 00/」を変換した結果を表している。縦方向に「反転」，横方向に「回転」の結果が見られる。

** 三方陣の基本図：反転形，90 度ずつの回転形 **

```
 00/          01/          02/          03/
   1  2  3      7  4  1      9  8  7      3  6  9
   4  5  6      8  5  2      6  5  4      2  5  8
   7  8  9      9  6  3      3  2  1      1  4  7
 10/          11/          12/          13/
   1  4  7      3  2  1      9  6  3      7  8  9
   2  5  8      6  5  4      8  5  2      4  5  6
   3  6  9      9  8  7      7  4  1      1  2  3
```

これらは，基本図をただ裏返したり転がしただけであるから，まったく別個の図になるわけでは決してない。本質的に同一の基本図の「異形」を表しているに過ぎない。

しかし，こうして８個を並べて見ると，原初三方陣の８個にそっくりではないか。いや，全く同じものではないか。

原初方陣には，変形まで一切を含めた「基本図の集合」としての意味があると言っても良いのではないかと思われる。

＃３．次に，ひな形とひな形変換法を８通りに変えて，各々が目的方陣の８個の解集合を果たして再構成できるかを調べる実験をする。これを一度は試みておかなければならない。

先ず原初方陣８個の解集合を用意しよう。これは全ての場合に共通だ。

** 共通の原初三方陣８個：これに対してひな形変換法を適用する **

```
      1/      2/      3/      4/      5/      6/      7/      8/
 ._._._. ._._._. ._._._. ._._._. ._._._. ._._._. ._._._. ._._._.
 |1|2|3| |1|4|7| |3|2|1| |3|6|9| |7|4|1| |7|8|9| |9|6|3| |9|8|7|
 |-+-+-| |-+-+-| |-+-+-| |-+-+-| |-+-+-| |-+-+-| |-+-+-| |-+-+-|
 |4|5|6| |2|5|8| |6|5|4| |2|5|8| |8|5|2| |4|5|6| |8|5|2| |6|5|4|
 |-+-+-| |-+-+-| |-+-+-| |-+-+-| |-+-+-| |-+-+-| |-+-+-| |-+-+-|
 |7|8|9| |3|6|9| |9|8|7| |1|4|7| |9|6|3| |1|2|3| |7|4|1| |3|2|1|
 '-'-'-' '-'-'-' '-'-'-' '-'-'-' '-'-'-' '-'-'-' '-'-'-' '-'-'-'
```

** ひな形とひな形変換法を８通りに変えて目的方陣を再構成する実験 **

```
//  void damt1(){
//    n[1]=m[4]; n[2]=m[9]; n[3]=m[2];
//    n[4]=m[3]; n[5]=m[5]; n[6]=m[7];
//    n[7]=m[8]; n[8]=m[1]; n[9]=m[6];
//  };
```

```
      1/      2/      3/      4/      5/      6/      7/      8/
 ._._._. ._._._. ._._._. ._._._. ._._._. ._._._. ._._._. ._._._.
 |4|9|2| |2|9|4| |6|7|2| |2|7|6| |8|3|4| |4|3|8| |8|1|6| |6|1|8|
 |-+-+-| |-+-+-| |-+-+-| |-+-+-| |-+-+-| |-+-+-| |-+-+-| |-+-+-|
 |3|5|7| |7|5|3| |1|5|9| |9|5|1| |1|5|9| |9|5|1| |3|5|7| |7|5|3|
 |-+-+-| |-+-+-| |-+-+-| |-+-+-| |-+-+-| |-+-+-| |-+-+-| |-+-+-|
 |8|1|6| |6|1|8| |8|3|4| |4|3|8| |6|7|2| |2|7|6| |4|9|2| |2|9|4|
 '-'-'-' '-'-'-' '-'-'-' '-'-'-' '-'-'-' '-'-'-' '-'-'-' '-'-'-'
 [Count = 8]  OK!
```

```
//  void damt2(){
//    n[1]=m[2]; n[2]=m[9]; n[3]=m[4];
//    n[4]=m[7]; n[5]=m[5]; n[6]=m[3];
//    n[7]=m[6]; n[8]=m[1]; n[9]=m[8];
//  };
```

```
       1/       2/       3/       4/       5/       6/       7/       8/
 ._._._.  ._._._.  ._._._.  ._._._.  ._._._.  ._._._.  ._._._.  ._._._.
 |2|9|4|  |4|9|2|  |2|7|6|  |6|7|2|  |4|3|8|  |8|3|4|  |6|1|8|  |8|1|6|
 |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|
 |7|5|3|  |3|5|7|  |9|5|1|  |1|5|9|  |9|5|1|  |1|5|9|  |7|5|3|  |3|5|7|
 |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|
 |6|1|8|  |8|1|6|  |4|3|8|  |8|3|4|  |2|7|6|  |6|7|2|  |2|9|4|  |4|9|2|
 '-'-'-'  '-'-'-'  '-'-'-'  '-'-'-'  '-'-'-'  '-'-'-'  '-'-'-'  '-'-'-'
 [Count = 8]  OK!
```

```
// void damt3(){
//   n[1]=m[6]; n[2]=m[7]; n[3]=m[2];
//   n[4]=m[1]; n[5]=m[5]; n[6]=m[9];
//   n[7]=m[8]; n[8]=m[3]; n[9]=m[4];
// };
```

```
       1/       2/       3/       4/       5/       6/       7/       8/
 ._._._.  ._._._.  ._._._.  ._._._.  ._._._.  ._._._.  ._._._.  ._._._.
 |6|7|2|  |8|3|4|  |4|9|2|  |8|1|6|  |2|9|4|  |6|1|8|  |2|7|6|  |4|3|8|
 |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|
 |1|5|9|  |1|5|9|  |3|5|7|  |3|5|7|  |7|5|3|  |7|5|3|  |9|5|1|  |9|5|1|
 |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|
 |8|3|4|  |6|7|2|  |8|1|6|  |4|9|2|  |6|1|8|  |2|9|4|  |4|3|8|  |2|7|6|
 '-'-'-'  '-'-'-'  '-'-'-'  '-'-'-'  '-'-'-'  '-'-'-'  '-'-'-'  '-'-'-'
 [Count = 8]  OK!
```

```
// void damt4(){
//   n[1]=m[2]; n[2]=m[7]; n[3]=m[6];
//   n[4]=m[9]; n[5]=m[5]; n[6]=m[1];
//   n[7]=m[4]; n[8]=m[3]; n[9]=m[8];
// };
```

```
       1/       2/       3/       4/       5/       6/       7/       8/
 ._._._.  ._._._.  ._._._.  ._._._.  ._._._.  ._._._.  ._._._.  ._._._.
 |2|7|6|  |4|3|8|  |2|9|4|  |6|1|8|  |4|9|2|  |8|1|6|  |6|7|2|  |8|3|4|
 |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|
 |9|5|1|  |9|5|1|  |7|5|3|  |7|5|3|  |3|5|7|  |3|5|7|  |1|5|9|  |1|5|9|
 |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|
 |4|3|8|  |2|7|6|  |6|1|8|  |2|9|4|  |8|1|6|  |4|9|2|  |8|3|4|  |6|7|2|
 '-'-'-'  '-'-'-'  '-'-'-'  '-'-'-'  '-'-'-'  '-'-'-'  '-'-'-'  '-'-'-'
 [Count = 8]  OK!
```

```
// void damt5(){
//   n[1]=m[8]; n[2]=m[3]; n[3]=m[4];
//   n[4]=m[1]; n[5]=m[5]; n[6]=m[9];
//   n[7]=m[6]; n[8]=m[7]; n[9]=m[2];
// };
```

```
       1/       2/       3/       4/       5/       6/       7/       8/
 ._._._.  ._._._.  ._._._.  ._._._.  ._._._.  ._._._.  ._._._.  ._._._.
 |8|3|4|  |6|7|2|  |8|1|6|  |4|9|2|  |6|1|8|  |2|9|4|  |4|3|8|  |2|7|6|
 |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|
 |1|5|9|  |1|5|9|  |3|5|7|  |3|5|7|  |7|5|3|  |7|5|3|  |9|5|1|  |9|5|1|
 |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|
 |6|7|2|  |8|3|4|  |4|9|2|  |8|1|6|  |2|9|4|  |6|1|8|  |2|7|6|  |4|3|8|
 '-'-'-'  '-'-'-'  '-'-'-'  '-'-'-'  '-'-'-'  '-'-'-'  '-'-'-'  '-'-'-'
 [Count = 8]  OK!
```

```
// void damt6(){
```

```
//    n[1]=m[4]; n[2]=m[3]; n[3]=m[8];
//    n[4]=m[9]; n[5]=m[5]; n[6]=m[1];
//    n[7]=m[2]; n[8]=m[7]; n[9]=m[6];
//  };
```

```
      1/       2/       3/       4/       5/       6/       7/       8/
 _ _ _    _ _ _    _ _ _    _ _ _    _ _ _    _ _ _    _ _ _    _ _ _
|4|3|8|  |2|7|6|  |6|1|8|  |2|9|4|  |8|1|6|  |4|9|2|  |8|3|4|  |6|7|2|
|-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|
|9|5|1|  |9|5|1|  |7|5|3|  |7|5|3|  |3|5|7|  |3|5|7|  |1|5|9|  |1|5|9|
|-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|
|2|7|6|  |4|3|8|  |2|9|4|  |6|1|8|  |4|9|2|  |8|1|6|  |6|7|2|  |8|3|4|
 - - -    - - -    - - -    - - -    - - -    - - -    - - -    - - -
```

```
[Count = 8]  OK!
```

```
//  void damt7(){
//    n[1]=m[8]; n[2]=m[1]; n[3]=m[6];
//    n[4]=m[3]; n[5]=m[5]; n[6]=m[7];
//    n[7]=m[4]; n[8]=m[9]; n[9]=m[2];
//  };
```

```
      1/       2/       3/       4/       5/       6/       7/       8/
 _ _ _    _ _ _    _ _ _    _ _ _    _ _ _    _ _ _    _ _ _    _ _ _
|8|1|6|  |6|1|8|  |8|3|4|  |4|3|8|  |6|7|2|  |2|7|6|  |4|9|2|  |2|9|4|
|-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|
|3|5|7|  |7|5|3|  |1|5|9|  |9|5|1|  |1|5|9|  |9|5|1|  |3|5|7|  |7|5|3|
|-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|
|4|9|2|  |2|9|4|  |6|7|2|  |2|7|6|  |8|3|4|  |4|3|8|  |8|1|6|  |6|1|8|
 - - -    - - -    - - -    - - -    - - -    - - -    - - -    - - -
```

```
[Count = 8]  OK!
```

```
//  void damt8(){
//    n[1]=m[6]; n[2]=m[1]; n[3]=m[8];
//    n[4]=m[7]; n[5]=m[5]; n[6]=m[3];
//    n[7]=m[2]; n[8]=m[9]; n[9]=m[4];
//  };
```

```
      1/       2/       3/       4/       5/       6/       7/       8/
 _ _ _    _ _ _    _ _ _    _ _ _    _ _ _    _ _ _    _ _ _    _ _ _
|6|1|8|  |8|1|6|  |4|3|8|  |8|3|4|  |2|7|6|  |6|7|2|  |2|9|4|  |4|9|2|
|-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|
|7|5|3|  |3|5|7|  |9|5|1|  |1|5|9|  |9|5|1|  |1|5|9|  |7|5|3|  |3|5|7|
|-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|  |-+-+-|
|2|9|4|  |4|9|2|  |2|7|6|  |6|7|2|  |4|3|8|  |8|3|4|  |6|1|8|  |8|1|6|
 - - -    - - -    - - -    - - -    - - -    - - -    - - -    - - -
```

```
[Count = 8]  OK!
```

```
** Listed by Kanji Setsuda on Dec.26, 2012 with MacOSX 10.8.2 and Xcode 4.5.2 **
```

出てくる順番こそ違え，目的方陣の解集合の内容は全く同じものが再現構成された。
かくして，どのひな形を用いても，同じ結果が得られることがわかった。
三方陣の場合には，このような美しい整合性があるのである。

「堂々巡り」の印象を持った方も居られるかもしれない。何故にこのような実験をしたかと言うと，ここで用いた手法・技術が，やがては高次方陣の研究時に役立つからなのだ。
高次方陣では，三方陣の時に見られた「整合性」が必ずしも見つかるわけではない。
ひな形を変えても全然変わらないケースもあるにはあるが，多くの場合に結果が異なってしまう。原初方陣そのものが変わって，同じ目的方陣を全数個再構成できないこともある。
面倒でも，対象ごとに一々そのチェックをしなければならないのである。

そして，極めて高い整合性のあるタイプを何とか発見して，報告しなければならない。
特に高次方陣では，原初方陣がなるべく共通・不変の独立したタイプが求められる。
それは，そうしたタイプが「根源は一つ」という貴重な美質を有する魔方陣の存在を暗示するからなのだ。
何とかそれを発見するために，今から準備しておかなければならないのである。

（初稿：2001 年；改定稿：2003，2006 年；改訂新稿：2012 年 12 月 26 日；
MacOSX 10.8.2 & Xcode 4.5.2 による作表・計数・執筆： 摂田 寛二(Kanji Setsuda)）

E-Mail Address: jag12001@nifty.com
Homepage Address: http://KanjiSetsuda.la.coocan.jp/

【追加報告：最新の研究成果とプログラム・リスト】

＃４．次に，最近の研究成果を反映させて，新しいリストを掲載する。2015 年版である。
先ず三方陣の原始解８個のリスト。内容は同じだが，出力順序を変更する。
五次や七次の対称方陣の研究でわかったことだが，n1＝１から始めるのでなく，中央の列の先頭の n2＝１から出力させるのが自然であると思うので，次表のように変更する。

```
 ** Primitive Solutions of Self-Complementary Magic Squares of Order 3 **
        1/         2/         3/         4/         5/         6/         7/         8/
 .--.--.--. .--.--.--. .--.--.--. .--.--.--. .--.--.--. .--.--.--. .--.--.--. .--.--.--.
 | 8| 1| 6| | 6| 1| 8| | 8| 3| 4| | 4| 3| 8| | 6| 7| 2| | 2| 7| 6| | 4| 9| 2| | 2| 9| 4|
 |--+--+--| |--+--+--| |--+--+--| |--+--+--| |--+--+--| |--+--+--| |--+--+--| |--+--+--|
 | 3| 5| 7| | 7| 5| 3| | 1| 5| 9| | 9| 5| 1| | 1| 5| 9| | 9| 5| 1| | 3| 5| 7| | 7| 5| 3|
 |--+--+--| |--+--+--| |--+--+--| |--+--+--| |--+--+--| |--+--+--| |--+--+--| |--+--+--|
 | 4| 9| 2| | 2| 9| 4| | 6| 7| 2| | 2| 7| 6| | 8| 3| 4| | 4| 3| 8| | 8| 1| 6| | 6| 1| 8|
 '--'--'--' '--'--'--' '--'--'--' '--'--'--' '--'--'--' '--'--'--' '--'--'--' '--'--'--'
```

第１番目のひな形が変わったので，原初方陣を求める条件式も次表のように変える。

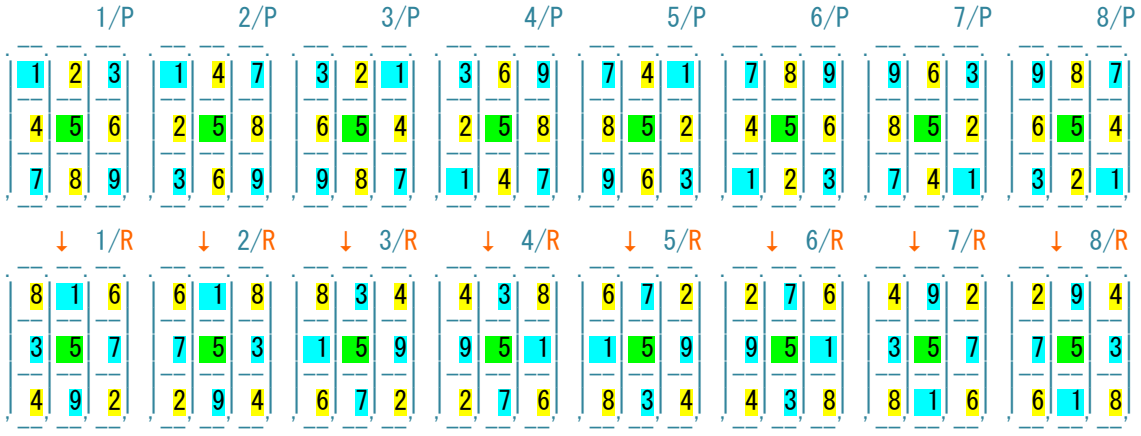
```
//
//* Make Prototype Squares of Order 3 *
//** Basic Form and Equations for Prototypes; **
//  .--.--.--.   n8+n1+n6=15;....(p1)
//  | 8| 1| 6|   n3+n5+n7=15;....(p2)
//  |--+--+--|   n4+n9+n2=15;....(p3)
//  | 3| 5| 7|   n8+n3+n4=15;....(p4)
//  |--+--+--|   n1+n5+n9=15;....(p5)
//  | 4| 9| 2|   n6+n7+n2=15;....(p6)
//  '--'--'--'   n8+n5+n2=15;....(p7)
//         and   n6+n5+n4=15;....(p8)
//** Self-Complementary Conditions: **
//  n1+n9=n2+n8=n3+n7=n4+n6=n5+n5=10;....(sc)
//
```

この連立方程式を用いて「逆関数」を解く要領で原初三方陣を求める。
そのプログラムは，巻末にリスト・アップしよう。
ひな形変換法も，次のように記述する。

```
//
//  void damt(){
//    n[1]=nm[8]; n[2]=nm[1]; n[3]=nm[6];
//    n[4]=nm[3]; n[5]=nm[5]; n[6]=nm[7];
//    n[7]=nm[4]; n[8]=nm[9]; n[9]=nm[2];
//  };
//
```

実行結果のリスト（2015 年版）は，次のようになった。

```
* Prototype Squares 3x3 and Reconstructions of S-C MS33
    by the 'Do-it-After-the-Model' Transformation: Type #1 *
  [Prototypes/P ↓ Reconstructions/R]
```




```
[Count = 8/8]    OK!

** Calculated and Listed by Kanji Setsuda on
**  Dec.14, 2015 with MacOSX EIC 10.11.2 & Xcode 7.2 **
```

{1, 3, 7, 9} の４数を青に彩色し「ユニット・ブルー」と名付ける。また {2, 4, 6, 8} の４数を黄に彩色し「ユニット・イェロー」と名付ける。中央の５は，中央の行列に属するだけでなく，主対角線上にもあるので，グリーンに彩色する。

これら２つのユニットは，原初方陣と再現構成解の間でどう行動しているであろうか？

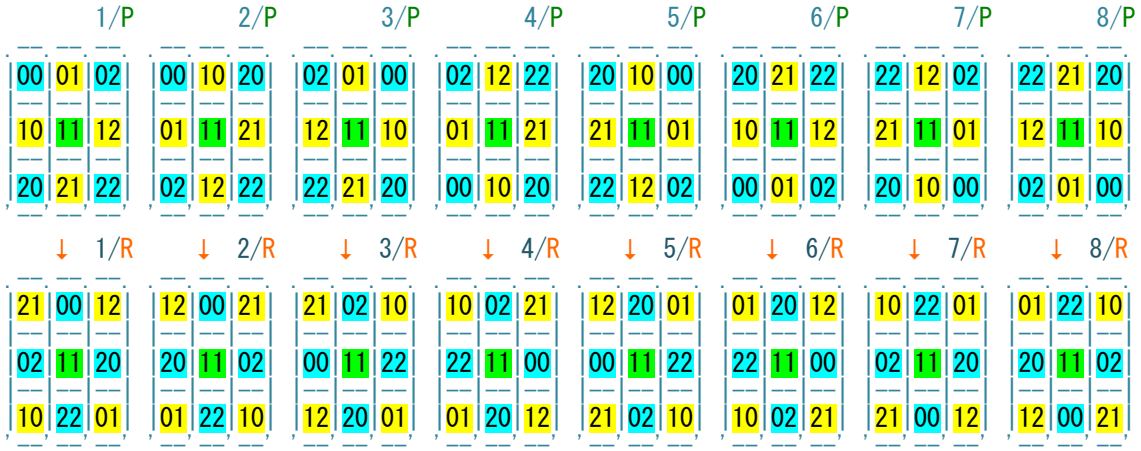
２つのユニットの配置に「相似性」や「相互交換」などの規則性が明快に示されることにお気づきだろうか。

こうした特徴は，高次方陣や立方体方陣でも見られるのである。

各種魔方陣の「内部構造」の更なる研究に役立つと考えられる。

三進法表記によるリストも示そう。

```
* Prototype Squares 3x3 and Reconstructions of S-C MS33
    by the 'Do-it-After-the-Model' Transformation: Type #1 *
  [Prototypes/P ↓ Reconstructions/R] in N3i
```




```
 [Count = 8/8] OK!

** Calculated and Listed by Kanji Setsuda on
**  Dec.16, 2015 with MacOSX EIC 10.11.2 & Xcode 7.2 **
```

E-Mail Address: jag12001@nifty.com
Homepage Address: http://KanjiSetsuda.la.coocan.jp/

[Program List]

```
//** 'Prototype Squares' and Self-Complementary MS33 **
//**  reconstructed by the 'DAM Transformation': Type 1; **
//** File: 'Proto33SC.c'; Dictated by Kanji Setsuda **
//**  on June 1, 2005 and Revised on Dec.14, 2015;  **
//**  with MacOSX EIC 10.11.2 & Xcode 7.2;  **
//
#include <stdio.h>
//
//* Global Variables *
short cnt, cnt2;
short LSM, CC;
short nm[10], uflg[10];
short mst[11][11], prt[11][11], obt[11][11];
//* Sub-Routines *
void stp01(void), stp02(void), stp03(void);
void stp04(void), stp05(void);
void msrecord(void);
//* Sub-Routines #2 *
void stp11(void), stp12(void), stp13(void);
void stp14(void), stp15(void);
void ansrecord(void);
void ansprint(void), recprint(void);
void oldn(void);
//
//** Main Program **
int main(){
 short n;
  printf("\n");
  printf("** 'Prototype Squares' of Order 3 and\n");
  printf("   Self-Complementary Magic Squares of Order 3 Reconstructed **\n");
  printf("\n");
  printf("* Primitive Solutions of Self-Complementary Magic Squares of Order 3 *\n");
  for(n=0;n<10;n++){nm[n]=0; uflg[n]=0; mst[0][n]=n;}
  LSM=15; CC=10; cnt=0;
  nm[5]=5; uflg[5]=1;    //* Set the n5=5 & the Used Flag *
  stp01();               //* Make the SC Magic Squares of Order 3 *
  uflg[5]=0; nm[5]=0;    //* Reset & Go to end *
 //
  for(n=0;n<10;n++){nm[n]=0; uflg[n]=0; prt[0][n]=n; obt[0][n]=n;}
  cnt2=0;
  nm[5]=5; uflg[5]=1;    //* Set the n5=5 & the Used Flag */
  stp11();               //* Make Prototype Squares and Transform */
  uflg[5]=0; nm[5]=0;    //* Reset & Go to end */
  oldn();
  ansprint();
  printf("\n");
  printf("* Prototype Squares 3x3 and Reconstructions of S-C MS33\n");
  printf("    by the 'Do-it-After-the-Model' Transformation: Type #1 *\n");
  recprint();
  printf(" [Count = %d/%d] OK!\n",cnt2,cnt);
  printf("\n");
  printf("** Calculated and Listed by Kanji Setsuda on\n");
  printf("**  Dec.14, 2015 with MacOSX EIC 10.11.2 & Xcode 7.2 **\n");
  printf("\n");
 return 0;
}
```

```
//
//   0/BF                   1/SC
//   .--.--.--.  n1+n2+n3=15;  .--.--.--.  8+1+6=15;
//   | 1| 2| 3|  n4+n5+n6=15;  | 8| 1| 6|  3+5+7=15;
//   |--+--+--|  n7+n8+n9=15;  |--+--+--|  4+9+2=15;
//   | 4| 5| 6|  n1+n4+n7=15;  | 3| 5| 7|  8+3+4=15;
//   |--+--+--|  n2+n5+n8=15;  |--+--+--|  1+5+9=15;
//   | 7| 8| 9|  n3+n6+n9=15;  | 4| 9| 2|  6+7+2=15;
//   '--'--'--'  n1+n5+n9=15;  '--'--'--'  8+5+2=15;
//         and  n3+n5+n7=15;        and  6+5+4=15;
//  n1+n9=n2+n8=n3+n7=n4+n6=n5+n5=10;
//
//* Set n2 & n8 *
void stp01(){
 short a,b;
  for(a=1;a<10;a++){b=CC-a;
   if((uflg[a]==0)&&(uflg[b]==0)){
     nm[2]=a; nm[8]=b;
     uflg[a]=1; uflg[b]=1;
     stp02();
     uflg[b]=0; uflg[a]=0;}
  }
}
//* Set n4 & n6 *
void stp02(){
 short a,b;
  for(a=1;a<10;a++){b=CC-a;
   if((uflg[a]==0)&&(uflg[b]==0)){
     nm[4]=a; nm[6]=b;
     uflg[a]=1; uflg[b]=1;
     stp03();
     uflg[b]=0; uflg[a]=0;}
  }
}
//* Set n1 & n9 *
void stp03(){
 short a,b;
  for(a=9;a>0;a--){b=CC-a;
   if((uflg[a]==0)&&(uflg[b]==0)){
     nm[1]=a; nm[9]=b;
     uflg[a]=1; uflg[b]=1;
     stp04();
     uflg[b]=0; uflg[a]=0;}
  }
}
//* Set n3=LSM-n1-n2 & n7 *
void stp04(){
 short a,b;
  a=LSM-nm[1]-nm[2];
  b=LSM-nm[9]-nm[8];
  if((0<a)&&(a<10)&&(a+b==CC)){
   if((uflg[a]==0)&&(uflg[b]==0)){
     nm[3]=a; nm[7]=b;
     uflg[a]=1; uflg[b]=1;
     stp05();
     uflg[b]=0; uflg[a]=0;}}
}
//* Check The Line-Sums *
void stp05(){
 short sm1,sm2,sm3;
  sm1=nm[1]+nm[4]+nm[7];
  sm2=nm[3]+nm[6]+nm[9];
  sm3=nm[3]+nm[5]+nm[7];
  if((sm1==LSM)&&(sm2==LSM)&&(sm3==LSM)){msrecord();}
```

```
}
//* Count The Answers *
void msrecord(){
 short n;
  cnt++;
  mst[cnt][0]=cnt;
  for(n=1;n<10;n++){mst[cnt][n]=nm[n];}
}
//
//* Make Prototype Squares of Order 3 *
//
//** Basic Form and Equations for Prototypes; **
//  .--.--.--.  n8+n1+n6=15;  ....(p1)
//  | 8| 1| 6|  n3+n5+n7=15;  ....(p2)
//  |--+--+--|  n4+n9+n2=15;  ....(p3)
//  | 3| 5| 7|  n8+n3+n4=15;  ....(p4)
//  |--+--+--|  n1+n5+n9=15;  ....(p5)
//  | 4| 9| 2|  n6+n7+n2=15;  ....(p6)
//  '--'--'--'  n8+n5+n2=15;  ....(p7)
//       and  n6+n5+n4=15;  ....(p8)
//** Self-Complementary Conditions: **
//  n1+n9=n2+n8=n3+n7=n4+n6=n5+n5=10   ....(sc)
//
//* Set n1 & n9 *
void stp11(){
 short a,b;
  for(a=1;a<10;a++){b=CC-a;
   if((uflg[a]==0)&&(uflg[b]==0)){
     nm[1]=a; nm[9]=b;
     uflg[a]=1; uflg[b]=1;
     stp12();
     uflg[b]=0; uflg[a]=0;}
  }
}
//* Set n3 & n7 *
void stp12(){
 short a,b;
  for(a=1;a<10;a++){b=CC-a;
   if((uflg[a]==0)&&(uflg[b]==0)){
     nm[3]=a; nm[7]=b;
     uflg[a]=1; uflg[b]=1;
     stp13();
     uflg[b]=0; uflg[a]=0;}
  }
}
//* Set n6 & n4 *
void stp13(){
 short a,b;
  for(a=9;a>0;a--){b=CC-a;
   if((uflg[a]==0)&&(uflg[b]==0)){
     nm[6]=a; nm[4]=b;
     uflg[a]=1; uflg[b]=1;
     stp14();
     uflg[b]=0; uflg[a]=0;}
  }
}
//* Set n8 & n2 *
void stp14(){
 short a,b;
  for(a=9;a>0;a--){b=CC-a;
   if((uflg[a]==0)&&(uflg[b]==0)){
     nm[8]=a; nm[2]=b;
     uflg[a]=1; uflg[b]=1;
     stp15();
```

```
      uflg[b]=0; uflg[a]=0;}
  }
}
//* Check The Line-Sum & of the Diagonals *
void stp15(){
 short sm1,sm2,sm3,sm4;
  sm1=nm[2]+nm[4]+nm[9];
  sm2=nm[3]+nm[4]+nm[8];
  sm3=nm[2]+nm[6]+nm[7];
  sm4=nm[1]+nm[6]+nm[8];
  if((sm1==LSM)&&(sm2==LSM)&&(sm3==LSM)&&(sm4==LSM)){
    ansrecord();
  }
}
//* Save The Answers and Transform after the Model: Type 1  *
void ansrecord(){
 short n;
  cnt2++;
  prt[cnt2][0]=cnt2;
  for(n=1;n<10;n++){prt[cnt2][n]=nm[n];}
 //* Do-it-After-the-Model Transformation: Type 1 *
  obt[cnt2][1]=nm[8]; obt[cnt2][2]=nm[1]; obt[cnt2][3]=nm[6];
  obt[cnt2][4]=nm[3]; obt[cnt2][5]=nm[5]; obt[cnt2][6]=nm[7];
  obt[cnt2][7]=nm[4]; obt[cnt2][8]=nm[9]; obt[cnt2][9]=nm[2];
}
//
//* Print The Answers *
void ansprint(){
 short l,m,m3,n;
  printf("");
  for(l=1;l<9;l++){printf("%10d/",mst[l][0]);}; printf("\n");
  printf("");
  for(l=1;l<9;l++){printf(" .--.--.--.");}; printf("\n");
  for(m=0;m<3;m++){m3=m*3;
    printf("");
    for(l=1;l<9;l++){
      printf(" |");
      for(n=1;n<4;n++){printf("%2d|",mst[l][m3+n]);}
    }
    printf("\n");
    if(m<2){for(l=1;l<9;l++){printf(" |--|--|--|");}}
    else{for(l=1;l<9;l++){printf(" '--'--'--'");}}
    printf("\n");
  }
}
//
//* Print The Prototypes and Reconstructions *
void recprint(){
 short l,m,m3,n;
  printf(" [Prototypes/P ↓ Reconstructions/R]\n");
  printf("");
  for(l=1;l<9;l++){printf("%9d/P",prt[l][0]);}; printf("\n");
  printf("");
  for(l=1;l<9;l++){printf(" .--.--.--.");}; printf("\n");
  for(m=0;m<3;m++){m3=m*3;
    printf("");
    for(l=1;l<9;l++){
      printf(" |");
      for(n=1;n<4;n++){printf("%2d|",prt[l][m3+n]);}
    }
    printf("\n");
    if(m<2){for(l=1;l<9;l++){printf(" |--|--|--|");}}
    else{for(l=1;l<9;l++){printf(" '--'--'--'");}}
    printf("\n");
```

```
  }
 //
  printf("");
  for(l=1;l<9;l++){printf("     ↓%3d/R",obt[l][0]);}; printf("\n");
  printf("");
  for(l=1;l<9;l++){printf(" .--.--.--.");}; printf("\n");
  for(m=0;m<3;m++){m3=m*3;
    printf("");
    for(l=1;l<9;l++){
      printf(" |");
      for(n=1;n<4;n++){printf("%2d|",obt[l][m3+n]);}
    }
    printf("\n");
    if(m<2){for(l=1;l<9;l++){printf(" |--|--|--|");}}
    else{for(l=1;l<9;l++){printf(" '--'--'--'");}}
    printf("\n");
  }
}
//
//* Find the Same Solution in the Old List *
void oldn(){
 short m,n,p,mtc;
  for(m=1;m<9;m++){
   for(n=1;n<9;n++){mtc=0;
     for(p=1;p<10;p++){if(obt[m][p]==mst[n][p]){mtc++;}else{break;}}
     if(mtc==9){obt[m][0]=mst[n][0]; break;}
   }
  }
}
//
```