第５章 ：原初および立体魔方陣の最新研究 ：摂田　寛二

第１節 ：原初および対称型立体三方陣の最新研究

＃０．この章では，「原初方陣」と「ひな形変換法」の応用問題として，三次，四次の立体魔方陣をとりあげる。魔方陣研究の新たな視点から三次元の立体方陣を見直したら，一体何が見えて来るであろうか。前に書いた諸章の諸節を前提にして更に研究を進めるので，特に原初方陣の項を御一読いただきたい。先ずは，立体三方陣から。

＃１．「目的方陣」を対称型の立体三方陣とする。
　下図のような３×３×３の三次元配列に，連続する自然数１ー27 を全個重複なく用いて構成した立体三方陣で，左右・上下・前後方向の９面で全行全列全柱の定和を実現し，かつ立体の主対角線４本の定和を実現する。必然的に「点対称型」になることが知られている。最初から対称型を目標に設計しても，得られる結果は同じである。
　解のリストを早速にも示そう。

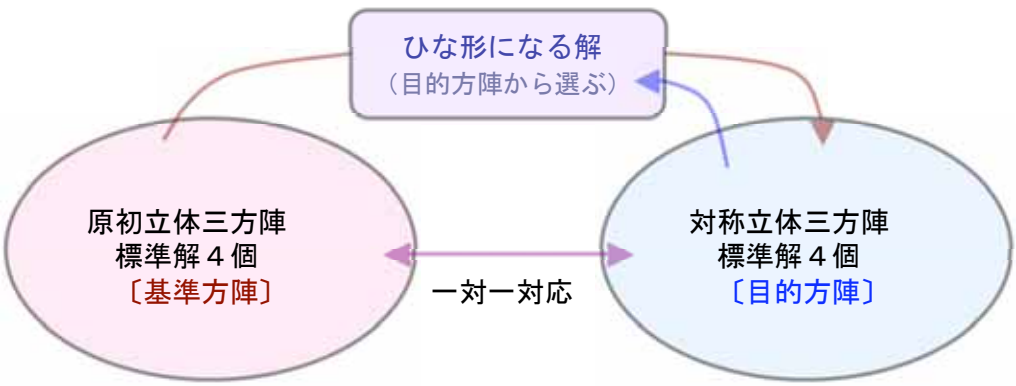
＊　対称立体三方陣の標準解のリスト（全）　＊

```
1/                          2/
  1-----15-----26             2-----15-----25
  |23      7     |12          |24      7     |11
  |  18-----20----- 4          |  16-----20----- 6
 17    | 19      6   |        18    | 19      5   |
  | 3 |   14    |25 |         | 1 |   14    |27 |
  |  22      9  |  11         |  23      9  |  10
 24---|- 8-----10   |        22---|- 8-----12   |
   16 |   21      5 |          17 |   21      4 |
      2-----13-----27             3-----13-----26

3/                          4/
  4-----17-----21             6-----16-----20
  |26      3     |13          |26      3     |13
  |  12-----22----- 8          |  10-----23----- 9
 18    | 19      5   |        17    | 21      4   |
  | 1 |   14    |27 |         | 1 |   14    |27 |
  |  23      9  |  10         |  24      7  |  11
 20---|- 6-----16   |        19---|- 5-----18   |
   15 |   25      2 |          15 |   25      2 |
      7-----11-----24             8-----12-----22   [Count = 4]  OK!
```

＃２．既に得られている解４個を「目的方陣」の解集合とする。そして，その中から１つを選んで「ひな形」とする。次に「基準方陣」となる「原初立体三方陣」４個を考える。

＊ 概念図 ＊　　ひな形変換法



そして，その２つの解集合の各要素の間に一対一対応の関係を確立したい。その架橋のために変換法を仮定し，それを「ひな形変換法」によって実現したい。

＃３．「原初方陣」とは，何であったか。
それは，規則的な配列を模倣した第一基本解と，それを変形した解の集合である。
最初の第一基本形だけは，両方陣とも既にわかっている。その第一基本形を下図のように並べて見ると，片や配列の基本図そのものであり，片や「ひな形」そのものである。

```
/原初方陣第一基本形          /目的方陣第一解
1-----10-----19             1-----15-----26
| 2     11    |20           |23     7     |12
|   3-----12-----21         |  18-----20----- 4
4   | 13     22   |        17   | 19      6   |
| 5 |   14    |23 |         | 3 |   14    |25 |
|   6     15  |  24         |  22     9   |  11
7---|-16-----25   |        24---|- 8-----10   |
  8 |   17     26 |          16 |   21      5 |
    9-----18-----27             2-----13-----27
```

＃４．原初方陣をいざ作る前に，先に「ひな形変換法」を説明しておこう。
それは，原初方陣から目的方陣に作り替える書き換え法のことであり，つまり構成法である。しかも，原初方陣のすべての解に対して，同じ変換法を実行することを想定している。たとえ，相手が４個であろうと何千個あろうと，常に同じひな形変換法を実行する。
具体的には，ひな形の解に即して次のような「変換の規則」を立て，それを実行する。

```
基準方陣の ｎ１  の値を目的方陣の  ｎ１  に入れる。
基準方陣の n23  の値を目的方陣の  ｎ２  に入れる。
基準方陣の n18  の値を目的方陣の  ｎ３  に入れる。
基準方陣の n17  の値を目的方陣の  ｎ４  に入れる。
基準方陣の ｎ３  の値を目的方陣の  ｎ５  に入れる。
  ............
基準方陣の ｎ５  の値を目的方陣の  n26  に入れる。
基準方陣の n27  の値を目的方陣の  n27  に入れる。
```

コンピュータ・プログラムに書くには，次のようなテーブル変換を用意するだけでよい。

```
/**/
void damt(void){
  tnm[1]=nm[1];   tnm[2]=nm[23];  tnm[3]=nm[18];
  tnm[4]=nm[17];  tnm[5]=nm[3];   tnm[6]=nm[22];
  tnm[7]=nm[24];  tnm[8]=nm[16];  tnm[9]=nm[2];
  tnm[10]=nm[15]; tnm[11]=nm[7];  tnm[12]=nm[20];
  tnm[13]=nm[19]; tnm[14]=nm[14]; tnm[15]=nm[9];
  tnm[16]=nm[8];  tnm[17]=nm[21]; tnm[18]=nm[13];
  tnm[19]=nm[26]; tnm[20]=nm[12]; tnm[21]=nm[4];
  tnm[22]=nm[6];  tnm[23]=nm[25]; tnm[24]=nm[11];
  tnm[25]=nm[10]; tnm[26]=nm[5];  tnm[27]=nm[27];
}
/**/
```

これを実行すると，原初方陣から目的方陣に書き変わる。変換法をそう作ったのだから，第１解は確かにそうなるのだが，はたして第２解以降もうまく行くだろうか。ぜひにもそうなるようにしたいのだが，そのために原初方陣の他の解をどうやって用意したら良いか？
この原初方陣の構成が，けっこう難問なのである。いつもさんざん苦労して来た。
次数を変え，タイプを変えると，原初方陣の作り方が大きく変わる。いろいろ実験をしてあれこれ試行錯誤し，悪戦苦闘して来たけれども，決まった必勝法が中々に見つからない。

どうも無いような感じだった。この何年もの間，何度も悩ましい思いをして来た。
　英国のオルレンショウ／ブラエ両先生が推奨する「クロス・サム等和」条件は確かに強力な方法だが，万事に有効というわけではない。目的方陣によっては，ほんの少数の原初方陣しか作らない。必ずしも一対一対応を見つけられない場合があるのである。

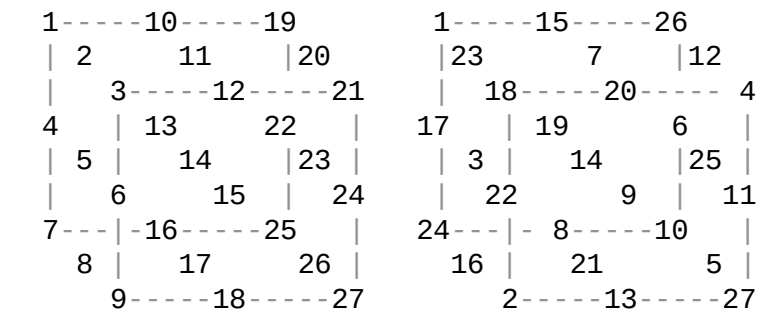
＃５．では，今回どうやって原初方陣の解集合を作るか。また試行錯誤するか。
　ここは，今までのやり方を変えて，新しい考え方での構成法をぜひ試してみたい。

　今は，目的方陣の解集合が既に得られている。ひな形の選定も，ひな形変換法も決まった。わかっていないのは，原初方陣の解集合である。ところが，それを作るのはひな形変換法の「逆の変換」に当たる。さかのぼって元の形を推理する方法を求めているのである。これを**「逆関数」を解く要領で試みてはどうか**。例えば，次のような基本図と定義式を用意する。

＊ 原初方陣のための基本定義式：行列柱の定和 ＊

```
n1+n23+n18=C;  | n1+n15+n26=C;  | n1+n17+n24=C;
n17+n3+n22=C;  | n23+n7+n12=C;  | n23+n3+n16=C;
n24+n16+n2=C;  | n18+n20+n4=C;  | n18+n22+n2=C;
n15+n7+n20=C;  | n17+n19+n6=C;  | n15+n19+n8=C;
n19+n14+n9=C;  | n3+n14+n25=C;  | n7+n14+n21=C;
n8+n21+n13=C;  | n22+n9+n11=C;  | n20+n9+n13=C;
n26+n12+n4=C;  | n24+n8+n10=C;  | n26+n6+n10=C;
n6+n25+n11=C;  | n16+n21+n5=C;  | n12+n25+n5=C;
n10+n5+n27=C;  | n2+n13+n27=C;  | n4+n11+n27=C;
```

＊ 基本図：原初方陣／　　／ひな形



＊ 原初方陣のための基本定義式：補数ペアの対称配置 ＊

```
n1+n27=n23+n5=n18+n10=n17+n11=n3+n25
  =n22+n6=n24+n4=n16+n12=n2+n26=n15+n13
  =n7+n21=n20+n8=n19+n9=n14+n14=CC
```

＊ リスト整形出力用の不等式群 ＊

```
n1<n18; n1<n24; n1<n2; n1<n26; n1<4; n1<n10; n1<n27; and n23>n17>n15;
```

　見てわかる通りに，今回は専ら「ひな形」に即して定義式を立てている。この連立方程式をいつものように解くのである。この条件を全部成り立たせる各変数の値の組を，求める解としてはどうか。手計算では大変なので，アシスタントのコンピュータに解かせる。

＃６．結果は大成功であった。めでたく原初方陣４個を構成した。その各個に「ひな形変換法」を実行したところ，目的方陣の対称立体三方陣の標準解４個を全数再構成した。

　こう書いてしまうと，余りの簡単さに読者も拍子抜けしてしまうに違いない。
　実は，コンピュータに与える命令書を書くプログラミングの作業が，けっこう大変で，神経を使う。拙作を文末に掲げるので，興味のある方はリストを読んでほしい。
　その前に実行結果を，解のリストにして報告しよう。

＊＊　原初立体三方陣と「ひな形変換法」によって再構成した　＊＊
＊＊　対称立体三方陣の標準解４個のリスト(全)：2007 年版　＊＊

1/原初

```
 1-----10-----19
 | 2     11    |20
 |   3-----12-----21
 4   | 13     22   |
 | 5 |   14    |23 |
 |   6     15  |  24
 7---|-16-----25   |
   8 |   17     26 |
     9-----18-----27
```

/目的

```
 1-----15-----26
 |23      7    |12
 |  18-----20----- 4
17   | 19      6   |
 | 3 |   14    |25 |
 |  22      9  |  11
24---|- 8-----10   |
  16 |   21     5  |
     2-----13-----27
```

2/原初

```
 2-----12-----19
 | 3     10    |20
 |   1-----11-----21
 6   | 13     23   |
 | 4 |   14    |24 |
 |   5     15  |  22
 7---|-17-----27   |
   8 |   18     25 |
     9-----16-----26
```

/目的

```
 2-----15-----25
 |24      7    |11
 |  16-----20----- 6
18   | 19      5   |
 | 1 |   14    |27 |
 |  23      9  |  10
22---|- 8-----12   |
  17 |   21     4  |
     3-----13-----26
```

3/原初

```
 4-----16-----19
 | 7     10    |22
 |   1-----13-----25
 8   | 11     23   |
 | 2 |   14    |26 |
 |   5     17  |  20
 3---|-15-----27   |
   6 |   18     21 |
     9-----12-----24
```

/目的

```
 4-----17-----21
 |26      3    |13
 |  12-----22----- 8
18   | 19      5   |
 | 1 |   14    |27 |
 |  23      9  |  10
20---|- 6-----16   |
  15 |   25     2  |
     7-----11-----24
```

4/原初

```
 6-----18-----21
 | 8     11    |23
 |   1-----13-----25
 9   | 12     24   |
 | 2 |   14    |26 |
 |   4     16  |  19
 3---|-15-----27   |
   5 |   17     20 |
     7-----10-----22
```

/目的

```
 6-----16-----20
 |26      3    |13
 |  10-----23----- 9
17   | 21     4   |
 | 1 |   14    |27 |
 |  24      7  |  11
19---|- 5-----18   |
  15 |   25     2  |
     8-----12-----22
```

[カウント = 4]　OK!

第２ー４解は，n5＝１になるタイプだが，第１解の n1＝１のタイプの「ひな形」を選んでも，きちんと再構成できている。従って，原初立体三方陣の解４個と，対称立体三方陣の解４個とが，このひな形変換法によって１対１に対応付けられることが，示せたと思う。

印象深いのは，原初方陣の４個だ。第一基本形からの変形の有様が，手に取るようにわかって面白い。これならば，構造研究の格好の素材になり得るかと思う。

＃７．ひな形を変えたらどうなるか。次の実験を試みなければなるまい。

選定した「ひな形」を唯一無二，絶対の存在と主張する人も居るが，それは原初方陣に特

別の条件を付けるからそう思えるので，そうしない場合にはどうなるか。実際にテストしてみなければ，わからない。他の解でも平等に「ひな形」になり得るものかどうか。

目的方陣の第２番解を新しいひな形に選び，その他の全てを次の如く変えてみた。

```
＊ 基本図：原初方陣／          ／ひな形-2
   1-----10-----19          2-----15-----25
   | 2     11    |20        |24     7     |11
   |   3-----12-----21      |  16-----20----- 6
   4   | 13     22   |     18   | 19      5   |
   | 5 |   14    |23 |     | 1 |   14    |27 |
   |   6     15  |  24     |  23     9   |  10
   7---|-16-----25   |     22---|- 8-----12   |
     8 |   17     26 |       17 |   21      4 |
       9-----18-----27          3-----13-----26
```

＊ 新しい「ひな形変換法」の規則 ＊

```
void damt(void){
  tnm[1]=nm[2];   tnm[2]=nm[24];  tnm[3]=nm[16];
  tnm[4]=nm[18];  tnm[5]=nm[1];   tnm[6]=nm[23];
  tnm[7]=nm[22];  tnm[8]=nm[17];  tnm[9]=nm[3];
  tnm[10]=nm[15]; tnm[11]=nm[7];  tnm[12]=nm[20];
  tnm[13]=nm[19]; tnm[14]=nm[14]; tnm[15]=nm[9];
  tnm[16]=nm[8];  tnm[17]=nm[21]; tnm[18]=nm[13];
  tnm[19]=nm[25]; tnm[20]=nm[11]; tnm[21]=nm[6];
  tnm[22]=nm[5];  tnm[23]=nm[27]; tnm[24]=nm[10];
  tnm[25]=nm[12]; tnm[26]=nm[4];  tnm[27]=nm[26];
}
```

＊ 原初方陣のための基本定義式-2：行列柱の定和 ＊

```
  n2+n24+n16=C;  | n2+n15+n25=C;  | n2+n18+n22=C;
  n18+n1+n23=C;  | n24+n7+n11=C;  | n24+n1+n17=C;
  n22+n17+n3=C;  | n16+n20+n6=C;  | n16+n23+n3=C;
  n15+n7+n20=C;  | n18+n19+n5=C;  | n15+n19+n8=C;
  n19+n14+n9=C;  | n1+n14+n27=C;  | n7+n14+n21=C;
  n8+n21+n13=C;  | n23+n9+n10=C;  | n20+n9+n13=C;
  n25+n11+n6=C;  | n22+n8+n12=C;  | n25+n5+n12=C;
  n5+n27+n10=C;  | n17+n21+n4=C;  | n11+n27+n4=C;
  n12+n4+n26=C;  | n3+n13+n26=C;  | n6+n10+n26=C;
```

＊ 原初方陣のための基本定義式-2：補数ペアの対称配置 ＊

```
  n2+n26=n24+n4=n16+n12=n18+n10=n1+n27
    =n23+n5=n22+n6=n17+n11=n3+n25=n15+n13
    =n7+n21=n20+n8=n19+n9=n14+n14=CC
```

＊ リスト整形出力用の不等式群-2 ＊

```
  n1<n7; n1<n9; n1<n19; n1<n21; n1<27; and n24>n23>n18>n17;
```

そして，逆関数を解く要領で，この連立方程式群を満たす解を求めた。当然のことではあるが，前とは違う原初方陣の解集合が得られた。ただし，第１解だけは当然同じであった。

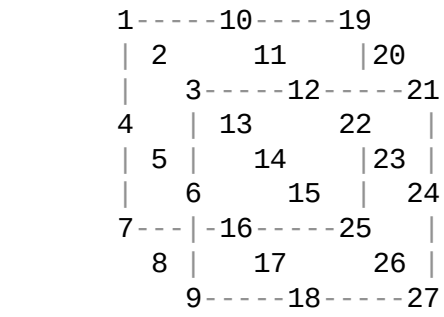
次のリストが，その実行結果である。ひな形変換したところ，目的方陣をすべて再構成した。n5＝１のタイプの解が３個続いた後で，最後に n1＝１のタイプの解を１個出力した。

ここでも，ひな形変換法による両方陣の間の一対一対応の関係が確かめられた。ひな形を変えても，まったく同じ結果が得られた。だから，４個の解の間には，ひな形候補の資格の面で優劣の差はないと，判断できる。（民主主義を，私は断然支持する！）

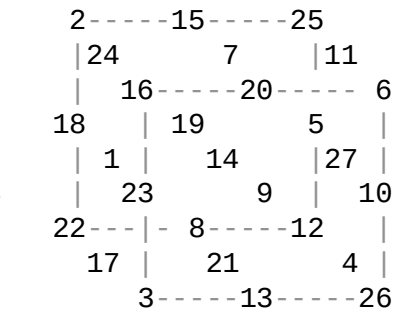
＊ 原初立体三方陣と別の「ひな形変換法」によって再構成した ＊

＊＊　対称立体三方陣の標準解４個のリスト(全)：2007 年版　＊＊

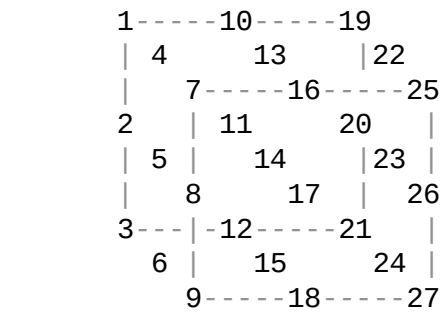
1/原初



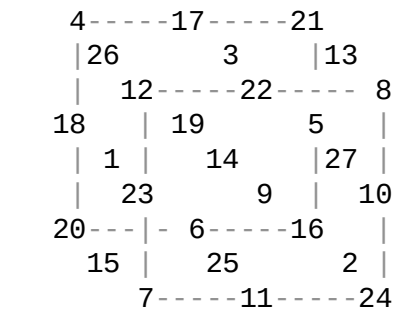
/目的-2



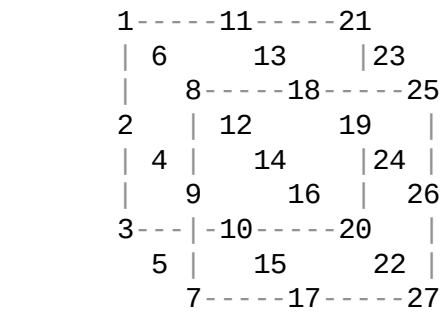
2/原初



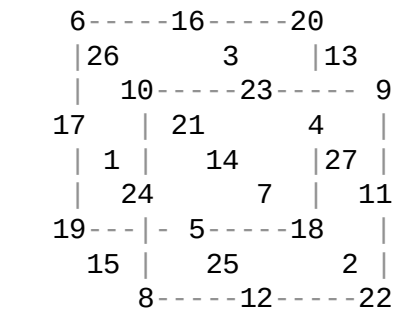
/目的-2



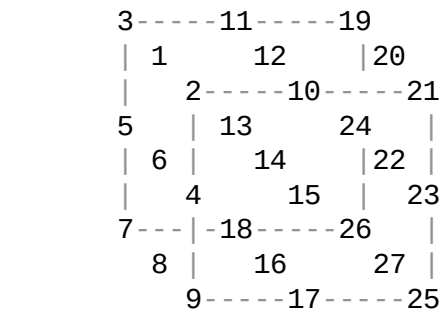
3/原初



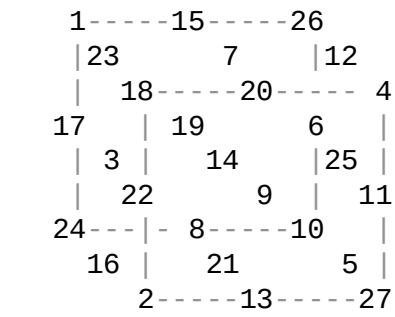
/目的-2



4/原初



/目的-2



[Count = 4]　OK!

その昔，初めて対称立体三方陣を調べた時には，n1＝１のタイプの解１個と，n5＝１のタイプの解３個とが，全然別の個性を持った方陣という印象を受けた。探索プログラムまで別にして調べた記憶がある。

しかし，今回は４つの解を全く同等の資格で統一的に扱うことができたように思う。そして，どの解も，真正の兄弟方陣であることがわかった。

解相互の内的・構造的な連関にも，強い興味を惹かれる。特に，原初方陣からいろいろな変換法が想像できて，新たな研究の発展を予感することができる。

＃８．逆関数を解く要領で原初方陣を構成する新しい方法は，平面方陣では何度か試みて成

功していたが，今回の立体方陣でもうまくいった。どうやら万能の必勝法になりそうだ。
　ただし，手順は必ずしも単純ではなく，万人向きではない恨みが残る。コンピュータに解かせる点でも，途中の大半が不透明で，かなり抵抗を感じる読者が居るかもしれない。
　しかし，それでも原初方陣がどうしても欲しい場面では，必要な方法である。否むしろ，これこそが普遍的で，真に望まれる方法であると，私は信じている。
　全てを「ひな形」そのものに教わろうとする我々の動機は，あくまで謙虚である。論拠不明の居丈高な独断論では決してない。およそ非難される謂れは無いと，信じている。

　それにしても，何故に「ひな形」は，一個の解の分際で全部を知っているのだろうか。原初方陣ばかりではない。目的方陣の中身さえ知っている。自分自身を変換法にすることで，原初方陣を作り，それを変換しながら仲間の目的方陣の全数を構成できてしまうのだから。「同語反復」に似た方法なのに，見事な結果を産み出すのだから，決して空論ではない。
　しかも，どの次数のどのタイプの魔方陣にも，必ず原初方陣・目的方陣・ひな形変換法を想定することができる。そして，美しい一対一対応を必ず発見することができる。
　一つの不思議が，次の不思議を生む。それ故に，探求の旅はとても終わりそうにない。

（初稿: on April 30, 2001, 執筆者：摂田　寛二(Kanji Setsuda);
改訂新稿: on March 6, 2007, with MacOSX & Xcode2.2: 摂田　寛二(Kanji Setsuda)）

E-Mail Address <jag12001@nifty.com>

＊ 付録：プログラム・リスト ＊

```
/** Prototype Cubes of Order 3 and Self-Complementary **/
/**  Magic Cubes Reconstructed by 'DAM Transformation **/
/** 'ProtoMC3.c' built by Kanji Setsuda **/
/** on Jan. 4, 2005 and Mar. 5, 2007 **/
/** Working with MacOSX and Xcode2.2 **/
/**/
#include <stdio.h>
/**/
short LSM, CC;
short cnt, cnt2;
short nm[28], uflg[28];
short tnm[28];
short anm[3][28];
/**/
void stp01(void), stp02(void), stp03(void), stp04(void);
void stp05(void), stp06(void), stp07(void), stp08(void);
void stp09(void), stp10(void), stp11(void);
void stp12(void), stp13(void), stp14(void);
void stp15(void), stp16(void);
void ansrecord(void), print2ans(void), print2sol(void);
void damt(void);
/**/
int main(){
 short n;
  printf("\n** Prototype Cubes of Order 3 and Self-Complementary **\n");
  printf("**  Magic Cubes Reconstructed by 'DAM Transformation **\n");
  for(n=0;n<28;n++){nm[n]=0; uflg[n]=0;}
  LSM=42; CC=28; cnt=0; cnt2=0;
  nm[14]=14; uflg[14]=1;
  stp01();
  uflg[14]=0;
  printf("[Count = %d]  OK!\n",cnt);
 return 0;
}
```

```
/* Begin The Search */
/* Set n1 & n27 & n1<n27 */
void stp01(){
 short n,m;
  for(n=1;n<14;n++){m=CC-n;
    if((uflg[n]==0)&&(uflg[m]==0)){
      uflg[n]=1; uflg[m]=1;
      nm[1]=n; nm[27]=m;
      stp02();
      uflg[m]=0; uflg[n]=0;}
   }
}
/* Set n26(>n1) & n2(>n1) */
void stp02(){
 short n,m;
  for(n=27;n>nm[1];n--){m=CC-n;
    if((m>nm[1])&&(uflg[n]==0)&&(uflg[m]==0)){
      uflg[n]=1; uflg[m]=1;
      nm[26]=n; nm[2]=m;
      stp03();
      uflg[m]=0; uflg[n]=0;}
   }
}
/* Set n15=LSM-n1-n26 & n13 */
void stp03(){
 short n,m;
  n=LSM-nm[1]-nm[26];
  m=LSM-nm[27]-nm[2];
  if((0<n)&&(n<28)&&(n+m==CC)){
    if((uflg[n]==0)&&(uflg[m]==0)){
      uflg[n]=1; uflg[m]=1;
      nm[15]=n; nm[13]=m;
      stp04();
      uflg[m]=0; uflg[n]=0;}}
}
/* Set n24(>n1) & n4(>n1) */
void stp04(){
 short n,m;
  for(n=27;n>nm[1];n--){m=CC-n;
    if((m>nm[1])&&(uflg[n]==0)&&(uflg[m]==0)){
      uflg[n]=1; uflg[m]=1;
      nm[24]=n; nm[4]=m;
      stp05();
      uflg[m]=0; uflg[n]=0;}
   }
}
/* Set n17=LSM-n1-n24 & n11 & n15<n17 */
void stp05(){
 short n,m;
  n=LSM-nm[1]-nm[24];
  m=LSM-nm[27]-nm[4];
  if((nm[15]<n)&&(n<28)&&(n+m==CC)){
    if((uflg[n]==0)&&(uflg[m]==0)){
      uflg[n]=1; uflg[m]=1;
      nm[17]=n; nm[11]=m;
      stp06();
      uflg[m]=0; uflg[n]=0;}}
}
/* Set n16=LSM-n2-n24 & n12 */
```

```
void stp06(){
 short n,m;
  n=LSM-nm[2]-nm[24];
  m=LSM-nm[26]-nm[4];
  if((0<n)&&(n<28)&&(n+m==CC)){
    if((uflg[n]==0)&&(uflg[m]==0)){
      uflg[n]=1; uflg[m]=1;
      nm[16]=n; nm[12]=m;
      stp07();
      uflg[m]=0; uflg[n]=0;}}
}
/* Set n23(>n17) & n5 */
void stp07(){
 short n,m;
  for(n=27;n>nm[17];n--){m=CC-n;
    if((uflg[n]==0)&&(uflg[m]==0)){
      uflg[n]=1; uflg[m]=1;
      nm[23]=n; nm[5]=m;
      stp08();
      uflg[m]=0; uflg[n]=0;}
   }
}
/* Set n18=LSM-n1-n23 & n18>n1 & n10(>n1) */
void stp08(){
 short n,m;
  n=LSM-nm[1]-nm[23];
  m=LSM-nm[27]-nm[5];
  if((nm[1]<n)&&(n<28)&&(n+m==CC)){
    if((nm[1]<m)&&(uflg[n]==0)&&(uflg[m]==0)){
      uflg[n]=1; uflg[m]=1;
      nm[18]=n; nm[10]=m;
      stp09();
      uflg[m]=0; uflg[n]=0;}}
}
/* Set n3=LSM-n23-n16 & n25 */
void stp09(){
 short n,m;
  n=LSM-nm[23]-nm[16];
  m=LSM-nm[5]-nm[12];
  if((0<n)&&(n<28)&&(n+m==CC)){
    if((uflg[n]==0)&&(uflg[m]==0)){
      uflg[n]=1; uflg[m]=1;
      nm[3]=n; nm[25]=m;
      stp10();
      uflg[m]=0; uflg[n]=0;}}
}
/* Set n22=LSM-n18-n2 & n6 */
void stp10(){
 short n,m;
  n=LSM-nm[18]-nm[2];
  m=LSM-nm[10]-nm[26];
  if((0<n)&&(n<28)&&(n+m==CC)){
    if((uflg[n]==0)&&(uflg[m]==0)){
      uflg[n]=1; uflg[m]=1;
      nm[22]=n; nm[6]=m;
      stp11();
      uflg[m]=0; uflg[n]=0;}}
}
/* Check(n17+n3+n22=LSM) */
```

```
void stp11(){
  if(nm[17]+nm[3]+nm[22]==LSM){stp12();}
}
/* Set n7 & n21 */
void stp12(){
 short n,m;
  n=LSM-nm[23]-nm[12];
  m=LSM-nm[5]-nm[16];
  if((0<n)&&(n<28)&&(n+m==CC)){
    if((uflg[n]==0)&&(uflg[m]==0)){
      uflg[n]=1; uflg[m]=1;
      nm[7]=n; nm[21]=m;
      stp13();
      uflg[m]=0; uflg[n]=0;}}
}
/* Set n20=LSM-n15-n7 & n8 */
void stp13(){
 short n,m;
  n=LSM-nm[15]-nm[7];
  m=LSM-nm[13]-nm[21];
  if((0<n)&&(n<28)&&(n+m==CC)){
    if((uflg[n]==0)&&(uflg[m]==0)){
      uflg[n]=1; uflg[m]=1;
      nm[20]=n; nm[8]=m;
      stp14();
      uflg[m]=0; uflg[n]=0;}}
}
/* Check(n3+n12+n21=42) */
void stp14(){
  if(nm[18]+nm[20]+nm[4]==LSM){stp15();}
}
/* Set n9=LSM-n20-n13 & n19 */
void stp15(){
 short n,m;
  n=LSM-nm[20]-nm[13];
  m=LSM-nm[8]-nm[15];
  if((0<n)&&(n<28)&&(n+m==CC)){
    if((uflg[n]==0)&&(uflg[m]==0)){
      uflg[n]=1; uflg[m]=1;
      nm[9]=n; nm[19]=m;
      stp16();
      uflg[m]=0; uflg[n]=0;}}
}
/* The Last Checks */
void stp16(){
 short sm1,sm2;
  sm1=nm[22]+nm[9]+nm[11];
  sm2=nm[17]+nm[19]+nm[6];
  if((sm1==LSM)&&(sm2==LSM)){ansrecord();}
}
/**/
/* Record the Answer */
void ansrecord(){
 short n;
  cnt++;
  anm[0][0]=cnt;
  for(n=1;n<28;n++){anm[0][n]=nm[n];}
  damt();
  for(n=1;n<28;n++){anm[1][n]=tnm[n];}
```

```
  print2sol();
}
/**/
/* Print 2 Answers */
void print2ans(){
 short l,l3,m,n;
  printf("%3d/ProtoT                /Object\n",anm[0][0]);
  for(l=0;l<9;l++){l3=l%3;
    for(n=0;n<l3;n++){printf("  ");}
    for(m=0;m<2;m++){
      if(m==0){printf("%4d%7d%7d     ",
        anm[m][l+1],anm[m][l+10],anm[m][l+19]);}
      else{printf("%4d%7d%7d",
        anm[m][l+1],anm[m][l+10],anm[m][l+19]);}
    }
    printf("\n");
  }
  printf("\n");
}
/**/
/* Print 2 Solutions */
void print2sol(){
  printf("%3d/ProtoT                /Object\n",anm[0][0]);
  printf("%4d-----%2d-----%2d%9d-----%2d-----%2d\n",
    anm[0][1],anm[0][10],anm[0][19],anm[1][1],anm[1][10],anm[1][19]);
  printf("    |%2d%7d    |%2d      |%2d%7d    |%2d\n",
    anm[0][2],anm[0][11],anm[0][20],anm[1][2],anm[1][11],anm[1][20]);
  printf("    |%4d-----%2d-----%2d    |%4d-----%2d-----%2d\n",
    anm[0][3],anm[0][12],anm[0][21],anm[1][3],anm[1][12],anm[1][21]);
  printf("%4d    | %2d%7d   |%5d   | %2d%7d   |\n",
    anm[0][4],anm[0][13],anm[0][22],anm[1][4],anm[1][13],anm[1][22]);
  printf("    |%2d |%5d    |%2d |    |%2d |%5d    |%2d |\n",
    anm[0][5],anm[0][14],anm[0][23],anm[1][5],anm[1][14],anm[1][23]);
  printf("    |%4d%7d  |%4d    |%4d%7d  |%4d\n",
    anm[0][6],anm[0][15],anm[0][24],anm[1][6],anm[1][15],anm[1][24]);
  printf("%4d---|-%2d-----%2d    |%5d---|-%2d-----%2d    |\n",
    anm[0][7],anm[0][16],anm[0][25],anm[1][7],anm[1][16],anm[1][25]);
  printf("%6d |%5d%7d |%7d |%5d%7d |\n",
    anm[0][8],anm[0][17],anm[0][26],anm[1][8],anm[1][17],anm[1][26]);
  printf("%8d-----%2d-----%2d%9d-----%2d-----%2d\n",
    anm[0][9],anm[0][18],anm[0][27],anm[1][9],anm[1][18],anm[1][27]);
  printf("\n");
}
/**/
/* 'Do-it-After-the-Model Transformation' */
void damt(void){
  tnm[1]=nm[1];   tnm[2]=nm[23];tnm[3]=nm[18];
  tnm[4]=nm[17];tnm[5]=nm[3];   tnm[6]=nm[22];
  tnm[7]=nm[24];tnm[8]=nm[16];tnm[9]=nm[2];
  tnm[10]=nm[15]; tnm[11]=nm[7];tnm[12]=nm[20];
  tnm[13]=nm[19]; tnm[14]=nm[14]; tnm[15]=nm[9];
  tnm[16]=nm[8];tnm[17]=nm[21]; tnm[18]=nm[13];
  tnm[19]=nm[26]; tnm[20]=nm[12]; tnm[21]=nm[4];
  tnm[22]=nm[6];tnm[23]=nm[25]; tnm[24]=nm[11];
  tnm[25]=nm[10]; tnm[26]=nm[5];tnm[27]=nm[27];
}
/**/
```