Chapter 5: Advanced Study of 'Prototype' Cubes: Kanji Setsuda

Section 1: 'Prototype' Cubes 3x3x3 and Self-Complementary Magic Cubes 3x3x3 Recomposed by 'DAMT'

#1. 'Prototype' Cubes of Order 3 and 'Do-it-After-the-Model Transformation'

Let's study further about 'Prototype' squares and cubes and also about 'Do-it-After-the-Model Transformation' by applying it to the actual cases of magic cubes.

Let's take the Self-complementary Magic Cubes of order 3 for our objects here.

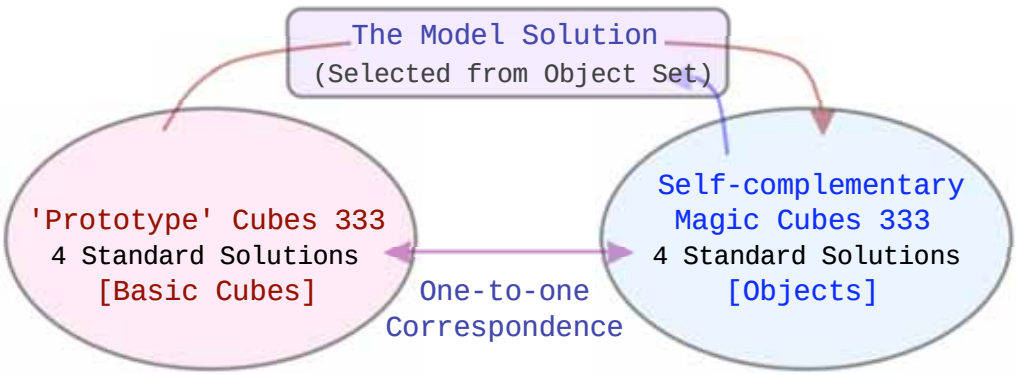
As you know, they are built in the three dimensional regular array 3x3x3 just like the solutions listed below. Every 3 elements on each row, each column and each pillar must add up to the same constant sum, and all 4 primary triagonals must also add up to 42, the same sum called 'magic constant'.

* Standard Solutions of **Self-Complementary Magic Cubes of Order 3** *

```
 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!
```

Each complementary pair of 28 {(1, 27), (2, 26), (3, 25), (4, 24), ..., (12, 16) or (13, 15)} should be put in the symmetrical positions with respect to the geometric center of the cube, where the number 14 should always be placed.

* Conceptual Diagram for **'Do-it-After-the-Model Transformation'** *



#2. The 'Model' Solution and the 'Do-it-After-the-Model Transformation'

Let's take those 4 self-complementary magic cubes of order 3 for our objects now, pick up one piece out of them for the 'Model' solution and define the rules of 'Do-it-After-the-Model Transformation' here.

I want to take the No.1 solution with n1=1 out of the list for our 'Model' and write the rules of 'DAM Transformation' now as follows:

```
   /Basic Form            /Object Model
   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
```

*** The rules of 'Do-it-After-the-Model Transformation': ***

```
  Take the value of n1  of Basic Cube  and put it into n1  of Object.
  Take the value of n23 of Basic Cube  and put it into n2  of Object.
  Take the value of n18 of Basic Cube  and put it into n3  of Object.
  Take the value of n17 of Basic Cube  and put it into n4  of Object.
  Take the value of n3  of Basic Cube  and put it into n5  of Object.
    . . . . . .
  Take the value of n5  of Basic Cube  and put it into n26 of Object.
  Take the value of n27 of Basic Cube  and put it into n27 of Object.
```

You may dictate the application 'DAMT' for your computer program as listed below. It is demonstrating the 'table exchange' of data of variables with their positions.

```
/**/
/* Application: 'DAM 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];}
/**/
```

We apply this DAM Transformation to our Prototype solutions one after another when we want to recompose our objects, self-complementary magic cubes of order 3.

We want to find the one-to-one correspondence between the two sets of solutions: Prototype cubes and object ones, making each bridge by our DAM Transformation.

#3. What are the 'Prototype Cubes' like? How can we make them?

We only know that the set of Prototype cubes should have as many solutions as our objects do and the first solution should be formed just like the three-dimensional regular array 3x3x3. The other three ones should be any variations of the basic form. But, we don't yet know anything precise about the real Prototype solutions.

How can we actually make all those Prototype solutions?

It has been one of the most difficult problems for us to compose such Prototype squares and cubes of any order or for any type. We often tried a lot and made many mistakes. We used to find the best way at last after a lot of tests and improvements.

We have not yet obtained any constant, universal way of composing Prototypes of any order or for any type. We know that even the powerful method using Equal Cross-sums is not always effective in other cases than the 'Most-Perfect' pan-magic squares of orders 4, 8, 12 and 16 recommended by Drs. Ollerenshaw and Brae.

I suppose it is the time when we have to make a whole conversion in our concept. It is now the chance when we adopt an entirely new method of composing Prototypes.

How about composing them by solving the 'inverse function' of DAMT?

Composing Prototypes from Objects is a reverse transformation of DAMT itself, for DAMT is originally designed for transforming each Prototype into Object. Therefore we have to find an 'inverse function' of DAMT to solve it and make Prototypes.

What can we imagine for that kind of inverse function and how can we solve it?

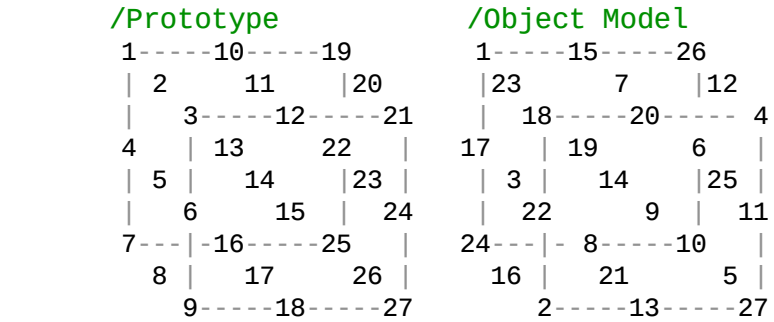
I would like to propose you the next process of doing our job as follows.

Prepare the basic forms and define the following new simultaneous equations:

** New **Definition for Prototype Cubes** of Order 3 **

* Basic Conditions for Prototype Cubes *

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;



* Self-Complementary Conditions *

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

* List-forming Inequality Conditions *

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

It is the very key-point that you have to assume every value of Object variables for every name of Prototype ones and define all the relationship among them as a self-complementary magic cube, tracing strictly after the Model solution itself.

Find every possible value that can satisfy all the simultaneous equations listed above, solving them one after another by your assistant personal computer.

I put the list of my computer program at the end of this article. Read it, please, if you are interested in it.

#4. Result Report of my recent Computation

```
*** Prototype Cubes of Order 3 and Self-Complementary ***
**  Magic Cubes Recomposed by our 'DAM Transformation' **
  ** 4 Standard Solutions of Prototypes and Objects **
  1/ProtoT                /Object
   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

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

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

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

As you see, we could compose all 4 Prototypes and recompose all 4 Objects by applying the single DAMT four times one after another. See the content of our list and compare it with the first list of object solutions, and you will find they are the same. We could really find the beautiful one-to-one correspondence between the two sets of Prototypes and Objects. Our new method proved to be really effective.

#5. How about taking another Model and another 'DAMT'?

Can we take another Model solution for the same purpose, make another DAMT, build all Prototypes and can we recompose all the same object solutions?

I feel we must make another experiment and test about it here.

I picked up a new Model solution, wrote new rules of DAM Transformation, defined new simultaneous equations, solved them and built new Prototype set.

```
* Basic Forms #2: *
  /ProtoT                /Model-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
```

```
* Set #2: Rules of 'DAM Transformation' *
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];
}
```

```
* Basic Conditions #2 for Prototype Cube: *
  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;
```

```
* Self-Complementary Conditions: *
  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
```

```
* List-forming Inequality Conditions #2: *
  n1<n7; n1<n9; n1<n19; n1<n21; n1<27; and n24>n23>n18>n17;
```

Finally I recomposed our object solutions by applying new DAMT to new Prototypes four times. Watch the next result list of my new computation, will you?

```
** Prototype Cubes of Order 3 and Self-Complementary **
** Magic Cubes Reconstructed by 'DAM Transformation' **
  1/ProtoT                /Object
   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

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

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

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

As you see, we have succeeded in making all Prototypes, remaking all Object cubes again and realizing the beautiful one-to-one correspondences between them.

It means another Model could work all right as well as the first Model could do.

Each of the four object solutions proves to be equal to the others, for it works all right as the best Model. It is amazing, isn't it?

#6. What does our success mean?

I am deeply impressed with the list of Prototype cubes every time when I see the result. What does the set of Prototype cubes mean? It looks like the set of Basic forms for our objects, including the most basic form and its variations. It must stimulate our study of transformation, for it implies how we can make the first one into others.

It is also amazing that we are taught everything by the Model solution.

(1) We are taught how to make the rules of 'DAM Transformation'.
(2) We are taught how to design the 'inverse function' of DAMT.
(3) We are taught how to make all Prototype cubes.
(4) We are taught how to recompose our object solutions.

What does this Model, our teacher, mean? What makes all those miracles possible?

I don't know much about it yet, but I am still concerned with the reason why the Model knows everything: the application 'DAMT', all Prototype solutions, all Object solutions and the beautiful one-to-one correspondence between them.

(First Version written in English on July 17, 2001;
Revised New Edition on March 10, 2007 with MacOSX and Xcode2.2)

by Kanji Setsuda: E-Mail Address <jag12001@nifty.com>

* Program List *

```
/** 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];
}
/**/
```