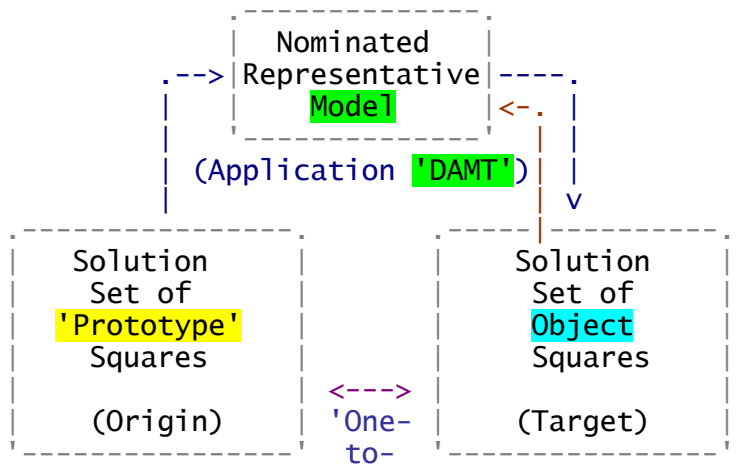
Part 4: New Advanced Study of Magic Squares and Cubes: Kanji Setsuda

Chapter 1: Fundamental Study of 'Prototype Squares' and 'Do-it-After-the-Model' Transformation

Section 1: Two Types of Squares of Order 3

#0. We are going to study about the two 'parallel worlds' of magic squares for some time and find the way how to build a beautiful 'bridge' over them.

** **Prototype Squares** and **'Do-it-After-the-Model' Transformation** **

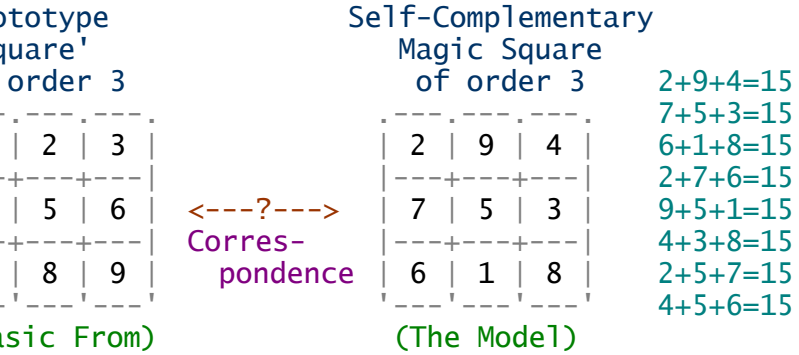


[**Figure 1**]

One is the world of Magic Squares of order 3, which is already familiar to us. The other is a new world that might make you feel something featureless at first, but you will surely feel something more natural and most powerful at last.

It is the world of 'Prototype Squares' of order 3.

Primary representative forms of them are shown as below.



[**Figure 2**: Primary Representative Forms]

The self-complementary magic square of order 3 is always the best example for you to enter the world of magic squares.

We already know that there are eight forms of 'primitive solutions' of this type, but they are essentially the same with one another, so that we can assume the 'standard solution' as only one. The other 7 forms should appear as the 'reflected patterns' or 'rotated patterns' of that only one solution.

[List 1: Eight Forms of Self-Complementary Magic Squares 3x3]

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

What do the 'Prototype Squares' of order 3 look like, then? The example of Figure 2 given above in p.1 seems so simple that it might look like the regular array of 3x3.

Yes. It is really the array itself. The set of prototype squares begins with this 'Basic Form' whose entries take their own values as the same with their names of positions. It is always the top, representative member of this group. The other 7 forms should appear as the reflected patterns or rotated ones of it, as you see in the next list below.

We are now going to study about these Prototype Squares at first and then find some correspondence between those two sets.

[List 2: The Set of 'Prototype Squares' of Order 3]

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

#1. What do you think of the property of these 'Prototype Squares'? Watch the eight forms in the list above and check each sum of 3 entries on any line available in it.

1+2+3=6, 4+5+6=15, 7+8+9=24, 1+4+7=12, 2+5+8=15, 3+6+9=18, 1+5+9=15, and 3+5+7=15

It seems to be neither a magic square, nor such a special square of any beautiful features that might have attracted us a lot to.

But the 4 equations: 1+5+9=15, 2+5+8=15, 3+5+7=15 and 4+5+6=15 are impressive. They are taken into 1+9=10, 2+8=10, 3+7=10, and 4+6=10. This means (1,9), (2,8), (3,7) and (4,6) are combined as the Complementary Pairs of 10, the main property of any Self-complementary square of order 3.

[**Figure 3**: Extended Form]

```
   .---.---.---.   Rows & Columns   Pan-diagonals
 3 | 1 | 2 | 3 | 1     1+2+3= 6;      1+5+9=15;
   |---+---+---|       4+5+6=15;      2+6+7=15;
 6 | 4 | 5 | 6 | 4     7+8+9=24;      3+4+8=15;
   |---+---+---|       1+4+7=12;      1+6+8=15;
 9 | 7 | 8 | 9 | 7     2+5+8=15;      2+4+9=15;
   '---'---'---'       3+6+9=18;      3+5+7=15;
* Complementary Pairs:
    1+9=10; 2+8=10; 3+7=10; 4+6=10; 5+5=10;
```

Figure 3 above shows each sum of 3 entries on each pan-diagonal is always equal to the magic constant 15. Yes, this is the same property with the one of any 'Pan-diagonal Magic Square'.

Although every row and every column does not always add up to the same sum, it is already the Simultaneous type: both Self-complementary and Pan-diagonal.

It is amazing, isn't it?

On top of that you can find some other interesting properties such as:

1+3=2+2, 4+6=5+5, 7+9=8+8, 1+7=4+4, 2+8=5+5, 3+9=6+6;
1+5=2+4, 2+6=3+5, 4+8=5+7, 5+9=6+8;
1+8=2+7, 2+9=3+8, 1+6=3+4, 4+9=6+7;

```
    1+9=3+7 (Equality of Cross-sums)
```

#2. Can you re-make all forms of 'prototype squares' with these properties?
Yes, you can.
Let's assume all the conditions listed as below now and make those squares and count them, shall we? This is the problem of permutations of the consecutive whole numbers from 1 to 9 in the array.

If you put neither a rank nor a determined order among n1, n2, n3, ..., and n9 (except n5), you will find next eight patterns of 'prototype squares' below.

[Figure 4: Basic Conditions of 'Prototype Squares' of Order 3]

```
            [Extended Form]
 n1+n9=10   3  1--2--3  1  2   n1+n5+n9=15 ...(pd1)
 n2+n8=10      |  |  |         n2+n6+n7=15 ...(pd2)
 n3+n7=10   6  4--5--6  4  5   n3+n4+n8=15 ...(pd3)
 n4+n6=10      |  |  |         n3+n5+n7=15 ...(pd4)
 n5+n5=10   9  7--8--9  7  8   n2+n4+n9=15 ...(pd5)
   (n5=5)                      n1+n6+n8=15 ...(pd6)
            3  1  2  3  1  2

 n1+n5=n2+n4, n2+n6=n3+n5, n4+n8=n5+n7, n5+n9=n6+n8;
 n1+n8=n2+n7, n2+n9=n3+n8, n1+n6=n3+n4, n4+n9=n6+n7;
 n1+n9=n3+n7 (Equality of Cross-sums)
 n1+n3=n2+n2, n4+n6=n5+n5, n7+n9=n8+n8,
 n1+n7=n4+n4, n2+n8=n5+n5, n3+n9=n6+n6;
 (The sum of outer 2 entries is equal to the sum of inner 2.)
```

[List 3: Eight Forms of 'Prototype Squares' of order 3]

```
    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
```

We have thus got all members of a new world. Let's compare these members with those eight self-complementary magic squares (**List 1**) each by each.

#3. Now I like to propose you a way of transforming each prototype square of **List 3** into each self-complementary magic square of **List 1**.
You may probably know of this way. It is well known as the 'Classical Composition' of a magic square of odd orders such as 3, 5 or 7.
It is also useful for you to transform one to the other, and next one to the other next, ..., and the last one to the other last.

[Figure 5: Transform it by the Classical Method of Composition]

```
 'Prototype Squares'  -->  Self-complementary Magic Squares

[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]                 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                   #

[3]                 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                   #

[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]                 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                   #

[6]                 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                   #

[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]                 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                   #
```

As you see, you could surely make all of our objects, Self-complementary type of Magic Squares of order 3, one after another up to the last.

It is amazing that each result is different from the others. Neither drop-out nor duplications occur among them, as you see.

This indicates there exists the 'one-to-one correspondence' between them. Thus those two solution-sets of different types of squares are combined so tight that you might imagine a 'bridge' is certainly built between these two 'parallel worlds'.

#4. Whenever you have the solution-set of 'prototype squares' and constantly apply this transformation method to every case, you can always get the same results.

You need not have got the solution set of object squares in advance any longer.

But why? What makes it possible? We can believe it is because there is any real structural concordance between those two worlds.

Let's come back to the first two patterns of Figure 2, and analyze them carefully.

(1) Complementary pairs of 10 are used in the object as well as in the original.

Properties/ /Type of squares	Every row and column add up to the same constant 15 ?	Every pan-diagonal add up to the same constant 15 ?
'Prototype Squares'	No	Yes
Self-Complementary Magic Squares	Yes	No

[Figure 6]

(2) Self-complementary Magic Squares have got the property that every row and column add up to same total, instead of losing the property that 3 entries on every pan-diagonal add up to the same constant. They are re-used and applied for the purpose of making the magic square.

(3) Why is each solution of both squares made into eight forms? It is because the same kind of reflections and rotations occur. Therefore we could think them for the eight faces of the only one 'standard solution'.

#5. Can you have any other method of transformation? Yes, you can.

Now I would like to propose you the newest method I call 'DAM Transformation'. Suppose such a method which makes the one into the other as below:

[Figure 7: New Rules of 'DAM Transformation']

```
'ProtoT      S-C       Put the value of n2(Original) into n1(Object).
 1 2 3      2 7 6      Put the value of n7(Original) into n2(Object).
 4 5 6 --> 9 5 1       Put the value of n6(Original) into n3(Object).
 7 8 9      4 3 8      Put the value of n9(Original) into n4(Object).
(Original)  (Object)   ......
                       Put the value of n3(Original) into n8(Object).
                       Put the value of n8(Original) into n9(Object).
```

This works all right. You can always make the objects anytime and anywhere you like, with these sets of 'transformation rules' and of eight 'prototype' squares.

[Prototype Squares of Order 3] **[List 4]**

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
↓	↓	↓	↓	↓	↓	↓	↓
1/	3/	2/	5/	4/	7/	6/	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

[Object Squares: SCMS33 Recomposed by 'DAMT' with the OldSN/]

The top pattern is always very important. Because it must be selected out of the solution set of self-complementary squares before all, for the purpose of making transformation rules. You must have got it first of all.

It is often called "a primary representative solution" or simply "the Model solution". It is expected to make the others by applying its transformation rules to prototypes.

I like to call this application '**D**o-it-**A**fter-the-**M**odel **T**ransformation'.

#6. Let me explain here about my newest method of composing prototype squares of order 3 and applying 'Do-it-After-the-Model Transformation' to them.

(1) First of all we must have got the Model Solution selected out of the solution set of object: Self-complementary Magic Squares of order 3.
(2) Define such conditions for Prototype Squares as listed below, after carefully consulting with the Model Solution.
(3) Compose the Prototype Squares under these conditions.
(4) Apply the 'DAM Transformation' to all Prototypes one after another, and get all the object solutions recomposed.

```
* Basic Forms and Conditions for 'Prototype Squares'
    /Prototype      /Object Model
    .--------.       .--------.
    | 1| 2| 3|       | 2| 7| 6|
    |--+--+--|       |--+--+--|
    | 4| 5| 6|       | 9| 5| 1|
    |--+--+--|       |--+--+--|
    | 7| 8| 9|       | 4| 3| 8|
    '--------'       '--------'
** New Basic Equations for Prototype Squares **
  n2+n9+n4=K....(b1)| n2+n7+n6=K....(b4)
  n7+n5+n3=K....(b2)| n9+n5+n1=K....(b5)
  n6+n1+n8=K....(b3)| n4+n3+n8=K....(b6)
** Primary Diagonals **
  n2+n5+n8=K....(b7)| n4+n5+n6=K....(b8)
** Self-Complementary Conditions: **
  n2+n8=n9+n1=n4+n6=n7+n3=n5+n5=C      ....(sc)
//
//* Do-it-After-the-Model Transformation *
void damt1(){
    tnm[1]=nm[2]; tnm[2]=nm[7]; tnm[3]=nm[6];
    tnm[4]=nm[9]; tnm[5]=nm[5]; tnm[6]=nm[1];
    tnm[7]=nm[4]; tnm[8]=nm[3]; tnm[9]=nm[8];
}
//                                    [Figure 8]
```

I would like you to read the program list of mine in 'C language' put at the end of this article and see how I do my job. The next list shows the recent result.

```
* 'Prototype Squares' and Self-Complementary Magic Squares of Order 3 *
[Goal: Self-complementary Magic Squares of Order 3]
     1/      2/      3/      4/      5/      6/      7/      8/
  2 7 6   2 9 4   4 3 8   4 9 2   6 1 8   6 7 2   8 1 6   8 3 4
  9 5 1   7 5 3   9 5 1   3 5 7   7 5 3   1 5 9   3 5 7   1 5 9
  4 3 8   6 1 8   2 7 6   8 1 6   2 9 4   8 3 4   4 9 2   6 7 2

[Prototype Squares of Order 3]
     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
```

```
[SCMS33 Recomposed by 'DAMT1' with Old SolN/]
     1/       3/       2/       5/       4/       7/       6/       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 = 8/8/8]
```

#7. Can you take another Model Solution, define it as listed below and apply the new 'DAM Transformation' with it? Yes, you can.

[Figure 9: Another Model and the new Rules of Transformation]

```
 'ProtoT      S-C        Put the value of n4(Original) into n1(Object).
  1 2 3      4 9 2       Put the value of n9(Original) into n2(Object).
  4 5 6 --> 3 5 7        Put the value of n2(Original) into n3(Object).
  7 8 9      8 1 6       Put the value of n3(Original) into n4(Object).
(Original)   (Object)     ......
                          Put the value of n1(Original) into n8(Object).
                          Put the value of n6(Original) into n9(Object).
//
//* Do-it-After-the-Model Transformation Type 2 *
void damt2(){
    tnm[1]=nm[4]; tnm[2]=nm[9]; tnm[3]=nm[2];
    tnm[4]=nm[3]; tnm[5]=nm[5]; tnm[6]=nm[7];
    tnm[7]=nm[8]; tnm[8]=nm[1]; tnm[9]=nm[6];
}
//
```

The next list shows all the results of my newest experiment.

```
 [Goal: Self-complementary Magic Squares of Order 3]
     1/       2/       3/       4/       5/       6/       7/       8/
  2 7 6    2 9 4    4 3 8    4 9 2    6 1 8    6 7 2    8 1 6    8 3 4
  9 5 1    7 5 3    9 5 1    3 5 7    7 5 3    1 5 9    3 5 7    1 5 9
  4 3 8    6 1 8    2 7 6    8 1 6    2 9 4    8 3 4    4 9 2    6 7 2

 [Prototype Squares of Order 3]
     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
    ↓        ↓        ↓        ↓        ↓        ↓        ↓        ↓
     4/       2/       6/       1/       8/       3/       7/       5/
  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
[SCMS33 Recomposed by 'DAMT2' with Old SolN/]

 [Count = 8/8/8]  OK!

** Recompiled and Listed by Kanji Setsuda on
    Mar.22, 2015 with MacOSX 10.10.2 & Xcode 6.2 **
```

You can take any form out of the eight Self-complementary objects, assume it as the Model Solution, define its rules of 'DAM Transformation', compose the set of

'Prototype Squares', and apply 'DAMT' to each prototype in order to recompose each object. Each of the eight Model Solutions can recompose the same set of objects.

There certainly exists the 'one-to-one correspondence' between the two sets, and you can imagine there are eight 'bridges' built over the two 'parallel worlds'.

(Written by Kanji Setsuda on Jul. 1, 2001; Revised on Oct.14, 2006;
Recompiled on Mar.22, 2015 with MacOSX 10.10.2 & Xcode 6.2;)

E-mail Address: jag12001@nifty.com

<< Program List >>

```
//** 'Prototype Squares' and Self-complementary Magic Squares of Order 3 **
//** File: 'ProtoT3SC.c' Dictated by Kanji Setsuda **
//**  on June 16, 2005 and Oct.12, 2006 with MacOSX and Xcode 2.2; **
//**  Recompiled on Mar.22, 2015 with MacOSX 10.10.2 & Xcode 6.2; **
//
#include <stdio.h>
//
//* Global Variables *
short cnt, cnt2;
short LSM, CC, damttp;
short nm[10], tnm[10], uflg[10];
short at[9][10], st[9][10], pt[9][10], tt[9][10];
//
//* Sub-Routines *
void stp01(void), stp02(void), stp03(void);
void stp04(void), stp05(void), stp06(void);
void msrecord(void);
//* Sub-Routines #2 *
void stp11(void), stp12(void), stp13(void);
void stp14(void), stp15(void), stp16(void);
void ansrecord(void);
void damt1(void), damt2(void);
void pr8ans(void);
void prtrns(void);
void oldn(void);
void sortp(void), exc(short x);
//
//** Main Program **
int main(){
 short m,n;
  printf("\n");
  printf("* 'Prototype Squares' and Self-Complementary Magic Squares 3x3 *\n");
  LSM=15; CC=10;
  for(n=0;n<10;n++){nm[n]=0; uflg[n]=0;}
  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
  printf("\n");
  printf("[Goal: Self-complementary Magic Squares of Order 3]\n");
  for(m=0;m<8;m++){for(n=0;n<10;n++){at[m][n]=st[m][n];}}
  pr8ans();
  //
  for(n=0;n<10;n++){nm[n]=0; uflg[n]=0;}
  cnt2=0;
  nm[5]=5; uflg[5]=1;    // Set the n5=5 & the Used Flag
  stp11();               // Make Prototype Squares and Transform
  sortp();               // Sort the Prototypes
  uflg[5]=0; nm[5]=0;    // Reset & Go to end
```

```
  //
  printf("\n");
  printf("[Prototype Squares of Order 3]\n");
  for(m=0;m<8;m++){for(n=0;n<10;n++){at[m][n]=pt[m][n];}}
  pr8ans();
  //
  damttp=1;     // Select Type of 'DAMT'
  damt1();
  oldn();
  printf("\n");
  printf("[SCMS33 Recomposed by 'DAMT%d' with Old SolN/]\n",damttp);
  for(m=0;m<8;m++){for(n=0;n<10;n++){at[m][n]=tt[m][n];}}
  pr8ans();
  damttp=2;     // Select Type of 'DAMT'
  damt2();
  oldn();
  printf("\n");
  printf("[SCMS33 Recomposed by 'DAMT%d' with Old SolN/]\n",damttp);
  for(m=0;m<8;m++){for(n=0;n<10;n++){at[m][n]=tt[m][n];}}
  pr8ans();
  printf("\n");
  //
  printf("** Transform it by the Classical Method of Composition **\n");
  oldn();
  prtrns();
  printf(" [Count = %d/%d/%d]  OK!\n",cnt,cnt2,cnt2);
  printf("** Recompiled and Listed by Kanji Setsuda on\n");
  printf("    Mar.22, 2015 with MacOSX 10.10.2 & Xcode 6.2 **\n");
  printf("\n");
 return 0;
}
//
//* Make the SC Magic Squares of Order 3 *
//  8  9  7  8  9  7  8    **  Basic Equations: **
//       .--------.           n1+n2+n3=K  ....(b1)
//  2  3| 1| 2| 3| 1  2      n4+n5+n6=K  ....(b2)
//       |--+--+--|           n7+n8+n9=K  ....(b3)
//  5  6| 4| 5| 6| 4  5      n1+n4+n7=K  ....(b4)
//       |--+--+--|           n2+n5+n8=K  ....(b5)
//  8  9| 7| 8| 9| 7  8      n3+n6+n9=K  ....(b6)
//       '--------'           n1+n5+n9=K  ....(b7)
//  2  3  1  2  3  1  2      n3+n5+n7=K  ....(b8)
//
//**  Self-Complementary Conditions: **
//n1+n9=n2+n8=n3+n7=n4+n6=n5+n5=C ....(sc)
//
//* Set n1 & n9 *
void stp01(){
 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;
     stp02();
     uflg[b]=0; uflg[a]=0;}
  }
}
//* Set n2 & n8 *
void stp02(){
 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;
     stp03();
```

```
      uflg[b]=0; uflg[a]=0;}
   }
}
//* Set n3=LSM-n1-n2 & n7 *
void stp03(){
 short a,b;
   a=LSM-nm[1]-nm[2]; b=CC-a;
   if((0<a)&&(a<10)){
    if((uflg[a]==0)&&(uflg[b]==0)){
      nm[3]=a; nm[7]=b;
      uflg[a]=1; uflg[b]=1;
      stp04();
      uflg[b]=0; uflg[a]=0;}}
}
//* Set n4=LSM-n1-n7 & n6 *
void stp04(){
 short a,b;
   a=LSM-nm[1]-nm[7]; b=CC-a;
   if((0<a)&&(a<10)){
    if((uflg[a]==0)&&(uflg[b]==0)){
      nm[4]=a; nm[6]=b;
      uflg[a]=1; uflg[b]=1;
      stp05();
      uflg[b]=0; uflg[a]=0;}}
}
//* Check The Line-Sums *
void stp05(){
 short sm1,sm2;
   sm1=nm[3]+nm[6]+nm[9];
   sm2=nm[7]+nm[8]+nm[9];
   if((sm1==LSM)&&(sm2==LSM)){msrecord();}
}
//* Count The Answers *
void msrecord(){
 short n;
   st[cnt][0]=cnt+1;
   for(n=1;n<10;n++){st[cnt][n]=nm[n];}
   cnt++;
}
//
//* Make 'Prototype Squares' *
//*    /Prototype      /Object Model
//   .--------.       .--------.
//   | 1| 2| 3|       | 2| 7| 6|
//   |--+--+--|       |--+--+--|
//   | 4| 5| 6|       | 9| 5| 1|
//   |--+--+--|       |--+--+--|
//   | 7| 8| 9|       | 4| 3| 8|
//   '--------'       '--------'
//** New Basic Equations for Prototype Squares **
//   n2+n9+n4=K  ....(b1) |  n2+n7+n6=K  ....(b4)
//   n7+n5+n3=K  ....(b2) |  n9+n5+n1=K  ....(b5)
//   n6+n1+n8=K  ....(b3) |  n4+n3+n8=K  ....(b6)
//** Primary Diagonals **
//   n2+n5+n8=K  ....(b7) |  n4+n5+n6=K  ....(b8)
//** Self-Complementary Conditions: **
//   n1+n9=n2+n8=n3+n7=n4+n6=n5+n5=C         ....(sc)
//
//** Make Prototype Squares and Transform **
//* Set n2 & n8 *
void stp11(){
 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;
      stp12();
      uflg[b]=0; uflg[a]=0;}
  }
}
//* Set n7 & n3 *
void stp12(){
 short a,b;
  for(a=1;a<10;a++){b=CC-a;
   if((uflg[a]==0)&&(uflg[b]==0)){
      nm[7]=a; nm[3]=b;
      uflg[a]=1; uflg[b]=1;
      stp13();
      uflg[b]=0; uflg[a]=0;}
  }
}
//* Set n6=LSM-n2-n7 & n4 *
void stp13(){
 short a,b;
  a=LSM-nm[2]-nm[7];
  if((0<a)&&(a<10)){
     b=LSM-nm[8]-nm[3];
     if(a+b==CC){
      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 n9=LSM-n2-n4 & n1 *
void stp14(){
 short a,b;
  a=LSM-nm[2]-nm[4];
  if((0<a)&&(a<10)){
     b=LSM-nm[8]-nm[6];
     if(a+b==CC){
      if((uflg[a]==0)&&(uflg[b]==0)){
         nm[9]=a; nm[1]=b;
         uflg[a]=1; uflg[b]=1;
         stp15();
         uflg[b]=0; uflg[a]=0;}}}
}
//* Check The Line-Sums *
void stp15(){
 short sm1,sm2;
  sm1=nm[7]+nm[5]+nm[3];
  sm2=nm[9]+nm[5]+nm[1];
  if((sm1==LSM)&&(sm2==LSM)){ansrecord();}
}
//* Save the Prototype *
void ansrecord(){
 short n;
  pt[cnt2][0]=cnt2+1;
  for(n=1;n<10;n++){pt[cnt2][n]=nm[n];}
  cnt2++;
}
//
//* Sort the Prototypes *
void sortp(){
 short d1,d2;
 short m,mx,f;
  mx=cnt2-1;
  do{f=0;
   for(m=0;m<mx;m++){
     d1=(pt[m][1]*9+pt[m][2])*9+pt[m][3];
```

```
      d2=(pt[m+1][1]*9+pt[m+1][2])*9+pt[m+1][3];
      if(d1>d2){exc(m); f=1;}
    }
    mx--;
   }while(f>0);
}
//
void exc(short x){
 short n,d;
  for(n=1;n<10;n++){
    d=pt[x][n]; pt[x][n]=pt[x+1][n]; pt[x+1][n]=d;
  }
}
//
//* Do-it-After-the-Model Transformation *
void damt1(){
 short m,n;
  for(m=0;m<cnt2;m++){
    for(n=1;n<=9;n++){nm[n]=pt[m][n];}
    tnm[1]=nm[2]; tnm[2]=nm[7]; tnm[3]=nm[6];
    tnm[4]=nm[9]; tnm[5]=nm[5]; tnm[6]=nm[1];
    tnm[7]=nm[4]; tnm[8]=nm[3]; tnm[9]=nm[8];
    for(n=1;n<=9;n++){tt[m][n]=tnm[n];}
  }
}
//
//* Do-it-After-the-Model Transformation Type 2 *
void damt2(){
 short m,n;
  for(m=0;m<cnt2;m++){
    for(n=1;n<=9;n++){nm[n]=pt[m][n];}
    tnm[1]=nm[4]; tnm[2]=nm[9]; tnm[3]=nm[2];
    tnm[4]=nm[3]; tnm[5]=nm[5]; tnm[6]=nm[7];
    tnm[7]=nm[8]; tnm[8]=nm[1]; tnm[9]=nm[6];
    for(n=1;n<=9;n++){tt[m][n]=tnm[n];}
  }
}
//
//* Find the Same Solution in the Old List *
void oldn(){
 short m,n,p,mtc;
  for(m=0;m<8;m++){
   for(n=0;n<8;n++){mtc=0;
     for(p=1;p<10;p++){if(tt[m][p]==st[n][p]){mtc++;}else{break;}}
     if(mtc==9){tt[m][0]=st[n][0]; break;}
   }
  }
}
//
//* Print The 8 Answers *
void pr8ans(){
 short l,m,l3,n;
  for(m=1;m<9;m++){
    printf("%6d/",at[m-1][0]);
    if(m<8){printf(" ");}
  }
  printf("\n");
  for(l=0;l<3;l++){l3=l*3;
    for(m=0;m<8;m++){
      printf(" ");
      for(n=1;n<4;n++){printf("%2d",at[m][l3+n]);}
      if(m<7){printf(" ");}
    }
    printf("\n");
  }
```

```
}
//
//** Transform it by the Classical Method of Composition **
//                 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             #
//
//* Print the Process of Classical Composition *
void prtrns(){
 short m,n;
 short p[10];
  for(m=0;m<cnt2;m++){
    for(n=0;n<10;n++){p[n]=pt[m][n];}
    printf("[%d]%15d              #\n",m+1,p[1]);
    printf("%4d%2d%2d%8d #%2d%10d%2d%2d%8d%2d%2d\n",
            p[1],p[2],p[3],p[4],p[2],p[4],p[9],p[2],p[4],p[9],p[2]);
    printf("%4d%2d%2d -->%2d #%2d #%2d --> #%2d%2d%2d # -->%2d%2d%2d\n",
            p[4],p[5],p[6],p[7],p[5],p[3],p[3],p[5],p[7],p[3],p[5],p[7]);
    printf("%4d%2d%2d%8d #%2d%10d%2d%2d%8d%2d%2d\n",
            p[7],p[8],p[9],p[8],p[6],p[8],p[1],p[6],p[8],p[1],p[6]);
    printf("%18d             #\n",p[9]);
    printf("\n");
  }
}
//
```