

Magic Square

A **magic square** of order n is an arrangement of n^2 numbers, usually distinct integers, in a square, such that the n numbers in all rows, all columns, and both diagonals sum to the same constant. A magic square contains the integers from 1 to n^2 .

The constant sum in every row, column and diagonal is called the **magic constant or magic sum**, M . The magic constant of a normal magic square depends only on n and has the following value:

$$M = n(n^2+1)/2$$

For normal magic squares of order $n = 3, 4, 5, \dots$, the magic constants are: 15, 34, 65, 111, 175, 260, ...

In this post, we will discuss how programmatically we can generate a magic square of size n . Before we go further, consider the below examples:

Magic Square of size 3

```
-----  
  2   7   6  
  9   5   1  
  4   3   8
```

Sum in each row & each column = $3*(3^2+1)/2 = 15$

Magic Square of size 5

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

Sum in each row & each column = $5*(5^2+1)/2 = 65$

Magic Square of size 7

```
-----  
 20  12   4  45  37  29  28  
 11   3  44  36  35  27  19  
  2  43  42  34  26  18  10  
 49  41  33  25  17   9   1  
 40  32  24  16   8   7  48  
 31  23  15  14   6  47  39
```

```
22 21 13 5 46 38 30
```

Sum in each row & each column = $7 \times (7^2 + 1) / 2 = 175$

Did you find any pattern in which the numbers are stored?

In any magic square, the first number i.e. 1 is stored at position $(n/2, n-1)$. Let this position be (i, j) . The next number is stored at position $(i-1, j+1)$ where we can consider each row & column as circular array i.e. they wrap around.

Three conditions hold:

1. The position of next number is calculated by decrementing row number of previous number by 1, and incrementing the column number of previous number by 1. At any time, if the calculated row position becomes -1, it will wrap around to $n-1$. Similarly, if the calculated column position becomes n , it will wrap around to 0.
2. If the magic square already contains a number at the calculated position, calculated column position will be decremented by 2, and calculated row position will be incremented by 1.
3. If the calculated row position is -1 & calculated column position is n , the new position would be: $(0, n-2)$.

Example:

Magic Square of size 3

```
-----
2  7  6
9  5  1
4  3  8
```

Steps:

1. position of number 1 = $(3/2, 3-1) = (1, 2)$
2. position of number 2 = $(1-1, 2+1) = (0, 0)$
3. position of number 3 = $(0-1, 0+1) = (3-1, 1) = (2, 1)$
4. position of number 4 = $(2-1, 1+1) = (1, 2)$
 Since, at this position, 1 is there. So, apply condition 2.
 new position = $(1+1, 2-2) = (2, 0)$
5. position of number 5 = $(2-1, 0+1) = (1, 1)$
6. position of number 6 = $(1-1, 1+1) = (0, 2)$
7. position of number 7 = $(0-1, 2+1) = (-1, 3)$ // this is tricky, see condition 3
 new position = $(0, 3-2) = (0, 1)$
8. position of number 8 = $(0-1, 1+1) = (-1, 2) = (2, 2)$ //wrap around
9. position of number 9 = $(2-1, 2+1) = (1, 3) = (1, 0)$ //wrap around

Based on the above approach, following is the working code:

```
#include<stdio.h>
```

```
#include<string.h>
```

```
// A function to generate odd sized magic squares
void generateSquare(int n)
{
    int magicSquare[n][n];

    // set all slots as 0
    memset(magicSquare, 0, sizeof(magicSquare));

    // Initialize position for 1
    int i = n/2;
    int j = n-1;

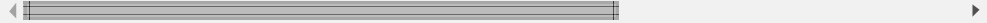
    // One by one put all values in magic square
    for (int num=1; num <= n*n; )
    {
        if (i== -1 && j==n) //3rd condition
        {
            j = n-2;
            i = 0;
        }
        else
        {
            //1st condition helper if next number goes to
            if (j == n)
                j = 0;
            //1st condition helper if next number is gone
            if (i < 0)
                i = n-1;
        }
        if (magicSquare[i][j]) //2nd condition
        {
            j -= 2;
            i++;
            continue;
        }
        else
            magicSquare[i][j] = num++; //set number

        j++; i--; //1st condition
    }

    // print magic square
    printf("The Magic Square for n=%d:\nSum of each row is ",
           n, n*(n*n+1)/2);
    for(i=0; i<n; i++)
    {
        for(j=0; j<n; j++)
            printf("%3d ", magicSquare[i][j]);
        printf("\n");
    }
}
```

```
// Driver program to test above function
```

```
int main()
{
    int n = 7; // Works only when n is odd
    generateSquare (n);
    return 0;
}
```



Output:

The Magic Square for n=7:

Sum of each row or column 175:

20	12	4	45	37	29	28
11	3	44	36	35	27	19
2	43	42	34	26	18	10
49	41	33	25	17	9	1
40	32	24	16	8	7	48
31	23	15	14	6	47	39
22	21	13	5	46	38	30

NOTE: This approach works only for odd values of n.

References:

http://en.wikipedia.org/wiki/Magic_square