CS1010E 16/17S1 Magic Square (Question)

Tags & Categories

Tags:

Categories:

Related Tutorials

Task Content

Magic Square

A magic square of order `n` is an arrangement of a sequence of n^2 numbers in a square grid such that the `n` numbers in all rows, columns and both diagonals sum to the same constant value. In particular, when filled in with consecutive values from `1` to `n`, this sum is given by $n(n^2 + 1)/2$.

An example of a magic square of order `3` is given below. Notice that all rows, columns and diagonals sum to `15`.

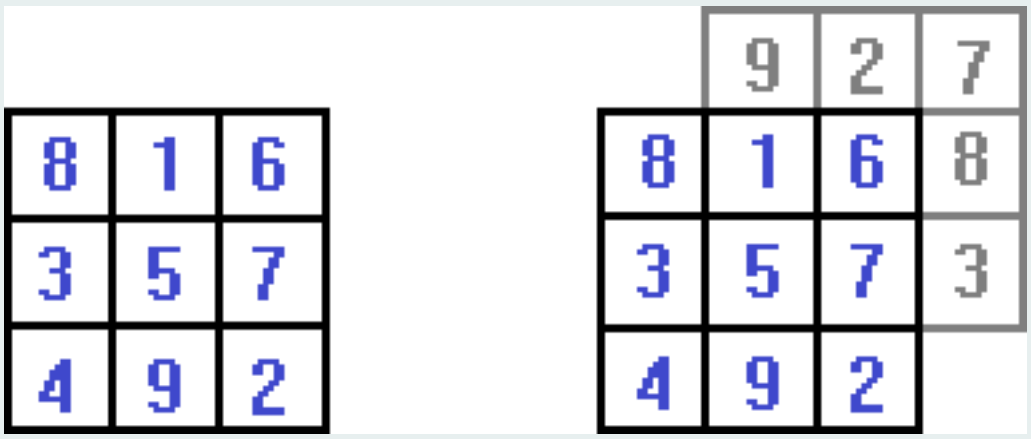
8	1	6
3	5	7
4	9	2

Given an order $n \geq 3$, there are three types of magic squares with different methods of construction.

Odd-order

When `n` is odd, start at the middle of the top row with the number `1`. Move diagonally (i.e up then right -- this is the only diagonal movement necessary) to the next cell with wrap-around (i.e. if the move leaves the grid, then wrap around to the other side). If this next cell is filled, then move down from the current cell. Fill the cell with the next number and repeat the step until the entire grid is filled.

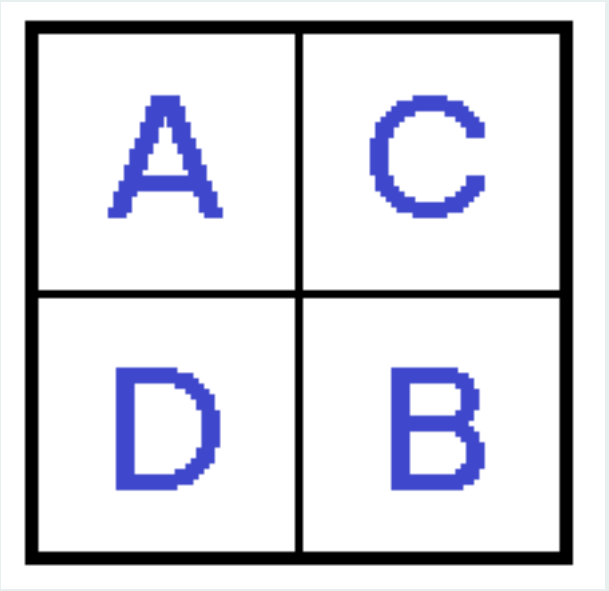
An example of a constructing the magic square of order `3` is given below.



- Place the number `1` at the central column of the first row.
- Since moving diagonally (up then right) lands outside the square, wrap around the top and bottom of the square to land at the bottom-right hand corner. Fill the cell with the next number `2`.
- Moving diagonally lands outside the square again. So wrap around the left and right of the square and fill in the number `3`.
- Notice that moving diagonally once more coincides with a filled cell `1`. So rather than move disgonally this time, move down instead and fill the cell with the number `4`.
- Using the diagonal moves, fill the next two numbers `5` and `6`.
- The next diagonal move is again outside the square. Moreover, wrapping around would coincide with a filled cell `4`. So move vertically down instead, and fill with the number `7`.
- Continue with two more diagonal movements, with wrap around, and fill the numbers `8` and `9`.

Singly-even order

When `n` is even and `n/2` is an odd number (e.g. `6, 10, 14,...`), construct four magic squares of order `n/2` (say A, B, C and D) with successively larger numbers, and lay them out in the following manner:



An example of the layout of four order `3` magic squares is shown below. Note the different starting numbers of the magic squares (in this case, `1`, `10`, `19` and `28`). Also note that this is not a magic square yet.

8	1	6	26	19	24
3	5	7	21	23	25
4	9	2	22	27	20
35	28	33	17	10	15
30	32	34	12	14	16
31	36	29	13	18	11

To generate the magic square of order `6`, the cells marked red and blue in the top and bottom halves have to be swapped.

35	1	6	26	19	24
3	32	7	21	23	25
31	9	2	22	27	20
8	28	33	17	10	15

30	5	34	12	14	16
4	36	29	13	18	11

Below are further examples of singly-even magic squares of order `10`, `14` and `30`. Observe that each of these magic squares is made up of four odd-ordered magic squares with red/blue regions, as well as purple/brown regions identified for subsequent swapping. Use the examples to identify and work out the general pattern of the respective swapping regions for progressively larger singly-even orders.

- Order `n = 10`

17	24	1	8	15	67	74	51	58	65
23	5	7	14	16	73	55	57	64	66
4	6	13	20	22	54	56	63	70	72
10	12	19	21	3	60	62	69	71	53
11	18	25	2	9	61	68	75	52	59
92	99	76	83	90	42	49	26	33	40
98	80	82	89	91	48	30	32	39	41
79	81	88	95	97	29	31	38	45	47
85	87	94	96	78	35	37	44	46	28
86	93	100	77	84	36	43	50	27	34

92	99
98	80
4	81
85	87
86	93
17	24
23	5
79	6
10	12
11	18

- Order `n = 14`

30	39	48	1	10	19	28	128	137	146	99	108	117	126
38	47	7	9	18	27	29	136	145	105	107	116	125	127
46	6	8	17	26	35	37	144	104	106	115	124	133	135
5	14	16	25	34	36	45	103	112	114	123	132	134	143
13	15	24	33	42	44	4	111	113	122	131	140	142	102
21	23	32	41	43	3	12	119	121	130	139	141	101	110
22	31	40	49	2	11	20	120	129	138	147	100	109	118
177	186	195	148	157	166	175	79	88	97	50	59	68	77
185	194	154	156	165	174	176	87	96	56	58	67	76	78
193	153	155	164	173	182	184	95	55	57	66	75	84	86

177	186	195	
185	194	154	
193	153	155	
5	161	163	
160	162	171	
168	170	179	
169	178	187	
30	39	48	
38	47	7	
46	6	8	

152	161	163	172	181	183	192	54	63	65	74	83	85	94		152	14	16	
160	162	171	180	189	191	151	62	64	73	82	91	93	53		13	15	24	
168	170	179	188	190	150	159	70	72	81	90	92	52	61		21	23	32	
169	178	187	196	149	158	167	71	80	89	98	51	60	69		22	31	40	

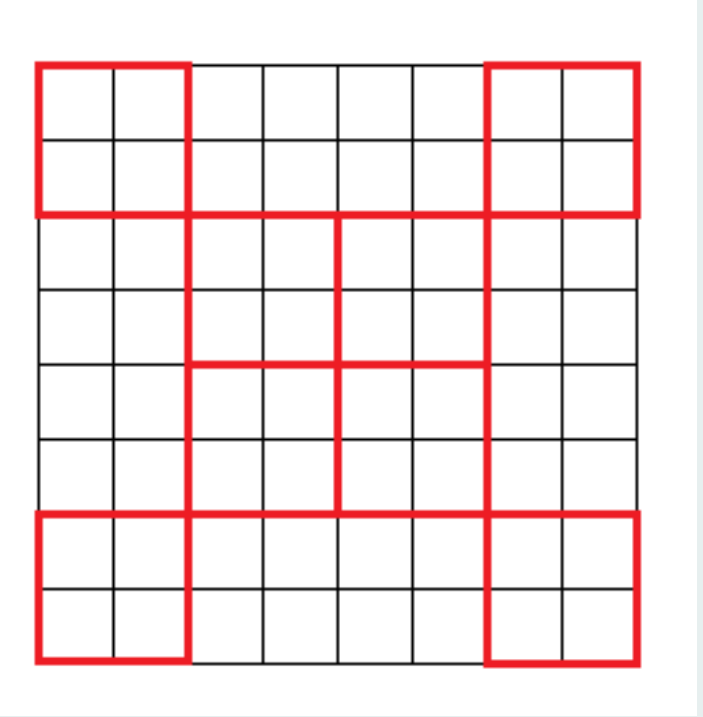
- Order `n = 30`

122	139	156	173	190	207	224	1	18	35	52	69	86	103	120	572	589	606	6
138	155	172	189	206	223	15	17	34	51	68	85	102	119	121	588	605	622	6
154	171	188	205	222	14	16	33	50	67	84	101	118	135	137	604	621	638	6
170	187	204	221	13	30	32	49	66	83	100	117	134	136	153	620	637	654	6
186	203	220	12	29	31	48	65	82	99	116	133	150	152	169	636	653	670	4
202	219	11	28	45	47	64	81	98	115	132	149	151	168	185	652	669	461	4
218	10	27	44	46	63	80	97	114	131	148	165	167	184	201	668	460	477	4
9	26	43	60	62	79	96	113	130	147	164	166	183	200	217	459	476	493	5
25	42	59	61	78	95	112	129	146	163	180	182	199	216	8	475	492	509	5
41	58	75	77	94	111	128	145	162	179	181	198	215	7	24	491	508	525	5
57	74	76	93	110	127	144	161	178	195	197	214	6	23	40	507	524	526	5
73	90	92	109	126	143	160	177	194	196	213	5	22	39	56	523	540	542	5
89	91	108	125	142	159	176	193	210	212	4	21	38	55	72	539	541	558	5
105	107	124	141	158	175	192	209	211	3	20	37	54	71	88	555	557	574	5
106	123	140	157	174	191	208	225	2	19	36	53	70	87	104	556	573	590	6
797	814	831	848	865	882	899	676	693	710	727	744	761	778	795	347	364	381	3
813	830	847	864	881	898	690	692	709	726	743	760	777	794	796	363	380	397	4
829	846	863	880	897	689	691	708	725	742	759	776	793	810	812	379	396	413	4
845	862	879	896	688	705	707	724	741	758	775	792	809	811	828	395	412	429	4
861	878	895	687	704	706	723	740	757	774	791	808	825	827	844	411	428	445	2
877	894	686	703	720	722	739	756	773	790	807	824	826	843	860	427	444	236	2
893	685	702	719	721	738	755	772	789	806	823	840	842	859	876	443	235	252	2
684	701	718	735	737	754	771	788	805	822	839	841	858	875	892	234	251	268	2
700	717	734	736	753	770	787	804	821	838	855	857	874	891	683	250	267	284	2
716	733	750	752	769	786	803	820	837	854	856	873	890	682	699	266	283	300	3
732	749	751	768	785	802	819	836	853	870	872	889	681	698	715	282	299	301	3

764	766	783	800	817	834	851	868	885	887	679	696	713	730	747	314	316	333	3
780	782	799	816	833	850	867	884	886	678	695	712	729	746	763	330	332	349	3
781	798	815	832	849	866	883	900	677	694	711	728	745	762	779	331	348	365	3

Doubly-even order

When both `n` and `n/2` are even numbers (e.g. `4, 8, 12, ...`), divide the entire magic square into `16` equal-sized and non-overlapping square sections and identify the `8` sections lying on the diagonals. This is illustrated below for constructing a magic square of order `8`.



By going through the grid in row-major order starting from the top-left, fill the numbers only if they are within the red regions; otherwise the numbers are skipped.

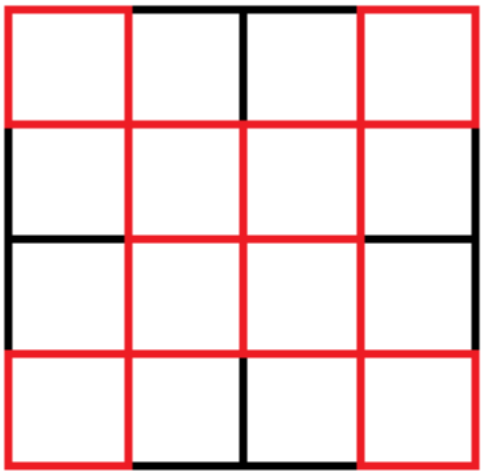
1	2					7	8
9	10					15	16
		19	20	21	22		
		27	28	29	30		
		35	36	37	38		
		43	44	45	46		
49	50					55	56
57	58					63	64

Now starting with the bottom-right of the grid, fill the rest of the numbers in reverse row-major order onto the unfilled regions.

1	2	62	61	60	59	7	8
9	10	54	53	52	51	15	16
48	47	19	20	21	22	42	41
40	39	27	28	29	30	34	33

32	31	35	36	37	38	26	25
24	23	43	44	45	46	18	17
49	50	14	13	12	11	55	56
57	58	6	5	4	3	63	64

Below is another example of constructing a doubly-even magic square of order `4`.



1			4
	6	7	
	10	11	
13			16

1	15	14
12	6	7
8	10	11
13	3	2

Task

Write a program that repeatedly reads an integer `n` and constructs the magic square of order `n (3 ≤ n ≤ 30)`.

The program terminates when `0` is read.

Take note of the following:

- Input to the program are only integers between `3` and `30` inclusive, or `0` to end the program.
- Use `printf("%4d",...)` to output each cell of the magic square.
- Adhere strictly to the instructions for constructing the magic squares; any other orientation of the magic square not in accordance to the instructions will be deemed as incorrect.
- You may write a function to check if the constructed magic square is valid.
- Only one sample test case is provided to test for format correctness. You should device your own test cases to test your program.

This task is divided into several levels. Read through all the levels (from first to last, then from last to first) to see how the different levels are related. **You may start from any level.**

Level 1

Name your program `magic1.c`

Write a program that repeatedly reads an integer `n` and outputs the value.

The program terminates when `0` is read.

The following is a sample run of the program. User input is underlined. Ensure that the last line of output is followed by

```
$ ./a.out
3
3
4
4
6
6
0
```

Click here to submit to CodeCrunch.

Check the correctness of the output by typing the following Unix command

Check the correctness of the output by typing the following Unix command

```
./a.out < magic.in | diff - magic1.out
```

To proceed to the next level (say level 2), copy your program by typing the Unix command

```
cp magic1.c magic2.c
```

Level 2

Name your program `magic2.c`

Write a program that repeatedly reads an integer `n`, outputs the value read and determines if this value is odd, singly-e

The program terminates when `0` is read.

The following is a sample run of the program. User input is underlined. Ensure that the last line of output is followed by

```
$ ./a.out
3
3 is odd
4
4 is doubly-even
6
6 is singly-even
0
```

Click here to submit to CodeCrunch.

Check the correctness of the output by typing the following Unix command

```
./a.out < magic.in | diff - magic2.out
```

To proceed to the next level (say level 3), copy your program by typing the Unix command

```
cp magic2.c magic3.c
```

Level 3

Name your program `magic3.c`

Write a program that repeatedly reads an integer `n`, outputs the value read and determines if this value is odd, singly-e odd, output the `n`-by-`n` grid with the starting value of `1` at the middle of the top row, and the rest of the values `0`.

The program terminates when `0` is read.

Hint: Remember to "zero" the values of the grid before generating the next magic square.

The following is a sample run of the program. User input is underlined. Ensure that the last line of output is followed by

```
$ ./a.out
3
3 is odd
   0   1   0
   0   0   0
   0   0   0
4
4 is doubly-even
5
5 is odd
   0   0   1   0   0
```

```
   0    0    0    0    0
   0    0    0    0    0
   0    0    0    0    0
   0    0    0    0    0
6
6 is singly-even
0
```

Click here to submit to CodeCrunch.

Check the correctness of the output by typing the following Unix command

```
./a.out < magic.in | diff - magic3.out
```

To proceed to the next level (say level 4), copy your program by typing the Unix command

```
cp magic3.c magic4.c
```

Level 4

Name your program `magic4.c`

Write a program that repeatedly reads an integer `n`, outputs the value read and determines if this value is odd, singly-e
odd, fill the `n`-by-`n` grid with the starting value of `1` at the middle of the top row, and proceed to move diagonally (i.e. up
around if necessary, to fill in successive values until a non-zero value is encountered. Output the grid.

The program terminates when `0` is read.

The following is a sample run of the program. User input is underlined. Ensure that the last line of output is followed by

```
$ ./a.out
3
3 is odd
   0    1    0
   3    0    0
   0    0    2
4
4 is doubly-even
5
5 is odd
   0    0    1    0    0
   0    5    0    0    0
   4    0    0    0    0
   0    0    0    0    3
   0    0    0    2    0
6
6 is singly-even
0
```

Click here to submit to CodeCrunch.

Check the correctness of the output by typing the following Unix command

```
./a.out < magic.in | diff - magic4.out
```

To proceed to the next level (say level 5), copy your program by typing the Unix command

```
cp magic4.c magic5.c
```

Level 5

Name your program `magic5.c`

Write a program that repeatedly reads an integer `n`, outputs the value read and determines if this value is odd, singly-e
odd, fill the `n`-by-`n` grid with the starting value of `1` at the middle of the top row, and proceed to move diagonally (i.e. up
around if necessary, to fill in successive values. If the next location is filled, then move down from the current location
should already be empty). Repeat the process until a magic square is obtained. Output the grid.

The program terminates when `0` is read.

The following is a sample run of the program. User input is underlined. Ensure that the last line of output is followed by

```
$ ./a.out
3
3 is odd
   8   1   6
   3   5   7
   4   9   2
4
4 is doubly-even
5
5 is odd
  17  24   1   8  15
  23   5   7  14  16
   4   6  13  20  22
  10  12  19  21   3
  11  18  25   2   9
6
6 is singly-even
0
```

Click here to submit to CodeCrunch.

Check the correctness of the output by typing the following Unix command

```
./a.out < magic.in | diff - magic5.out
```

To proceed to the next level (say level 6), copy your program by typing the Unix command

```
cp magic5.c magic6.c
```

Level 6

Name your program `magic6.c`

Write a program that repeatedly reads an integer `n`, outputs the value read and determines if this value is odd, singly-e
odd, construct and output the magic square accordingly. On the other hand, if `n` is singly-even, construct and output th
`B, C, D`) each of odd order `n/2` with successive starting values as given in the problem description above.

The program terminates when `0` is read.

The following is a sample run of the program. User input is underlined. Ensure that the last line of output is followed by

```
$ ./a.out
3
3 is odd
   8   1   6
   3   5   7
   4   9   2
4
4 is doubly-even
6
6 is singly-even
A
   8   1   6
```

```
   3    5    7
   4    9    2
B
  17   10   15
  12   14   16
  13   18   11
C
  26   19   24
  21   23   25
  22   27   20
D
  35   28   33
  30   32   34
  31   36   29
0
```

Click here to submit to CodeCrunch.

Check the correctness of the output by typing the following Unix command

```
./a.out < magic.in | diff - magic6.out
```

To proceed to the next level (say level 7), copy your program by typing the Unix command

```
cp magic6.c magic7.c
```

Level 7

Name your program `magic7.c`

Write a program that repeatedly reads an integer `n`, outputs the value read and determines if this value is odd, singly-e
odd, construct and output the magic square accordingly. On the other hand, if `n` is singly-even, construct the four mag
each of odd order `n/2` with successive starting values and lay them out appropriately as given in the problem descript
not a valid magic square yet.

Hint: You may want to define a function to copy values from one square grid to another, but at a specified location. As

`void copy(int grid1[][MAX], int n, int grid2[][MAX], int r, int c);`
Copy `n x n` elements of `grid1` to `grid2`, starting at the top-left location `(r, c)` of `grid2`.

The program terminates when `0` is read.

The following is a sample run of the program. User input is underlined. Ensure that the last line of output is followed by

```
$ ./a.out
3
3 is odd
   8    1    6
   3    5    7
   4    9    2
4
4 is doubly-even
6
6 is singly-even
   8    1    6   26   19   24
   3    5    7   21   23   25
   4    9    2   22   27   20
  35   28   33   17   10   15
  30   32   34   12   14   16
  31   36   29   13   18   11
```

Click here to submit to CodeCrunch.

Check the correctness of the output by typing the following Unix command

```
./a.out < magic.in | diff - magic7.out
```

To proceed to the next level (say level 8), copy your program by typing the Unix command

```
cp magic7.c magic8.c
```

Level 8

Name your program `magic8.c`

Write a program that repeatedly reads an integer `n`, outputs the value read and whether the value is odd, singly-even d construct and output the magic square accordingly. On the other hand, if `n` is singly-even, construct the four magic squ of odd order `n/2` with successive starting values and lay them out appropriately as shown in the problem description a the red/blue and purple/brown regions and perform the necessary swaps so as to produce a valid magic square.

Hint: You may want to define a function to facilitate swapping of the different regions within the same grid. As an exam

`void swap(int grid[][MAX], int rows, int cols, int r1, int c1, int r2, int c2);`
Swaps two rectangular regions of size `rows x cols`, both within `grid`, one of which is located with the top-le other located with the top-left at `(r2, c2)`.

The program terminates when `0` is read.

The following is a sample run of the program. User input is underlined. Ensure that the last line of output is followed by

```
$ ./a.out
3
3 is odd
   8   1   6
   3   5   7
   4   9   2
4
4 is doubly-even
6
6 is singly-even
  35   1   6  26  19  24
   3  32   7  21  23  25
  31   9   2  22  27  20
   8  28  33  17  10  15
  30   5  34  12  14  16
   4  36  29  13  18  11
```

Click here to submit to CodeCrunch.

Check the correctness of the output by typing the following Unix command

```
./a.out < magic.in | diff - magic8.out
```

To proceed to the next level (say level 9), copy your program by typing the Unix command

```
cp magic8.c magic9.c
```

Level 9

Name your program `magic9.c`

Write a program that repeatedly reads an integer `n`, determines whether the value is odd, singly-even or doubly-even, square accordingly.

The program terminates when `0` is read.

The following is a sample run of the program. User input is underlined. Ensure that the last line of output is followed by

```
$ ./a.out
3
   8   1   6
   3   5   7
   4   9   2
4
   1  15  14   4
  12   6   7   9
   8  10  11   5
  13   3   2  16
6
  35   1   6  26  19  24
   3  32   7  21  23  25
  31   9   2  22  27  20
   8  28  33  17  10  15
  30   5  34  12  14  16
   4  36  29  13  18  11
8
   1   2  62  61  60  59   7   8
   9  10  54  53  52  51  15  16
  48  47  19  20  21  22  42  41
  40  39  27  28  29  30  34  33
  32  31  35  36  37  38  26  25
  24  23  43  44  45  46  18  17
  49  50  14  13  12  11  55  56
  57  58   6   5   4   3  63  64
```

Click here to submit to CodeCrunch.

Check the correctness of the output by typing the following Unix command

```
./a.out < magic.in | diff - magic9.out
```