A3213

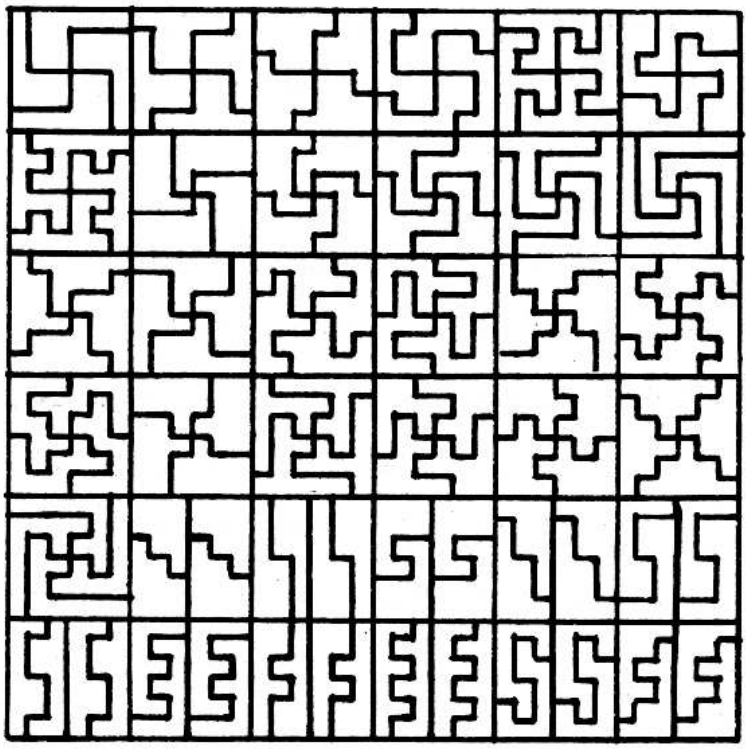
Parkin

Problem 15 called for the number of ways in which an 8 x 8 checkerboard could be cut into four congruent pieces, cutting along the lines of the checkerboard. The cover of issue No. 7 showed the 37 unique ways this could be done for a 6 x 6 board.

Thomas R. Parkin, Vice President, Control Data Corporation, using an algorithm given below implemented on a CDC 6600, has extended the known results as follows:

board size	number of ways
2	1
4	5
6	37
8	782
10	44240

with an estimate for the 12 x 12 board of 7750000. Mr. Parkin estimates that the 12 x 12 case, using his algorithm on the CDC STAR, would take 5 to 10 minutes, and the 14 x 14 case from 30 to 60 hours. The following is reproduced from his letter.



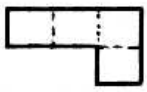
PC15-4

It might be well, first, to show some comparison numbers to support the seemingly large values for C{N}. We observe that each shape which cuts the square checkerboard into four congruent pieces is a polyomino of order N^2. Therefore, we might suspect that the number C{N} would be less than the corresponding number of polyominoes of the corresponding number of squares, since, of course, there are poloyominoes of any given size {≥4} which cannot be used to divide the square array into four congruent pieces, i.e., the 1 x 16 shape. Let's look at a comparative table:

n	1	2	3	4	5	6
N=2n	2	4	6	8	10	12
N^2	4	16	36	64	100	144
$N^2/4=M$	1	4	9	16	25	36
P{M}	1	5	1285	13079255	$\approx 2.03\times10^{12}$	$\approx 5\times10^{18}$
C{N}	1	5	37	782	44240	$\approx 7.75\times10^{6}$

Thus we see that, in general, $C\{N\} \ll P\{N^2\}$.

On the other hand, one might argue that even though all the $P\{N^2\}$ cannot be used, one or more might be used several times;

for example, the  piece for N=4 can be used two different

ways to yield two countably distinct divisions of the board into four congruent pieces. {Incidentally, if I were setting the rules, I would exclude this case since the stated problem is "to cut the square checkerboard into four congruent pieces", and once these pieces have been cut, I maintain their origin is no longer germane; however, this is not a problem except when $N \equiv 0 \bmod 4$ and it can be argued that the emphasis is on 'cut', and not the shape of the cut piece, so I won't press the point.} Even so, an examination of the counts of $P\{N^2\}$ by symmetry type also shows that pieces possessing both 90° rotational symmetry at one point and 180° symmetry at another are vanishingly few, thus the conclusion, $C\{N\} \ll P\{N^2\}$ seems valid.

There are a few observations about the problem which are necessary before one can understand the algorithm which I used for counting the possible cases. First of all, it is necessary to adopt a systematic procedure for enumerating the shapes, and, as a precursor to development of that systematic procedure, it

is necessary to adopt some taxonometric scheme for classifying the shapes. Note that there are two classes of cuts which will divide the checkerboard into congruent pieces, and these may be classified as those which cut the square array into four pieces such that the edges of the pieces are 90° rotationally symmetric within the square, and the remaining cases where the square is divided into two rectangles by a line through the center, bisecting two sides, and then the rectangle is divided into two congruent pieces by a line which is 180° rotationally symmetric. Next, note that the pieces may be displayed either obverse or reverse, i.e., mirror images, or they may all be oriented in some canonical fashion.

In my experience with writing programs to deal with two dimensional shapes, I have found that coding the shapes is quite tricky and difficult to work with; however, dealing with the edges of shapes is sometimes simpler. Thus, in this problem I chose to consider the square array of 2N x 2N squares in the plane to be a {2N+1} x {2N+1} square array of lattice points in the plane and trace paths corresponding to the edges of the shapes of interest.

The interior edges of the various pieces then provide for a taxonomy and allow a systematic procedure for generating shapes. Note that every piece in the square case involves an edge which traces a path from the outer perimeter of the square lattice to the center point. Furthermore, this path originates, say, along the upper left border of the square down to the center line, and from no other place. {This is the canonical orientation.} It is these unique paths which connect the border with the center which we enumerate {in the square case} and similarly, with suitable restrictions, for the rectangular case.

We note, of course, that the origin of the paths in the square case ranges over only one-eighth of the border {because of rotation and reflection symmetry}, and over only one quarter of the border in the rectangular case.

Now, of course, the algorithm is simple. Start at each appropriate border point and trace all possible unique paths to the appropriate center. I realize that this statement is descriptive but hardly constructive, so I will further detail the process.

1. Start at a corner, proceed down one side {or across the top}, and select the next point which has not been used as a starting point. Stop after using the center line point {or less than or equal to one-quarter of the top}. After selecting the border point, mark all other border points as "disallowed". Enter the border point in a list at Level 1.

2. Select the next lattice point inside the array, i.e., toward the center, and enter this point at Level 2. Note that there is only one possible choice for the second point on a path, given the first point on the border.

3. Mark the four {or two} symmetrical points to the last selected point as "disallowed". {The purpose of this disallowance is to preclude any point from appearing on more than one path, an obvious impossibility, thus, as each point on a path is selected, we simply check off the corresponding symmetrical points which are then no longer available for extensions of the current path.}

4. Add to the list, at the current level, the coordinates of the 0, 1, 2, or 3 possible points which could possibly be the extension of this path and mark them "unselected". Note that of the four points surrounding a given point, one of them is where the path came from, so that there are, at most, three possible extensions. Furthermore, it is possible to trace a path into a cul-de-sac such that no further extension of that path is possible and the path has not arrived at the center, hence the 0 possibility.

5. If there are further eligible selections at this level, select the next unselected eligible extension point on this path at this level, mark it "selected", and enter it at the next level. If there are no further selections at this level, remove the "disallowed" exclusions for the selected point, and the selected point, at this level. Back up one level. If Level 1 has been reached, all done this border point; if not, repeat this Step 5.

6. Match the selected point against the end point {or points} to see if this path is finished. If not, recursively repeat Steps 3, 4, 5, and 6 until the path terminates or an exit at Step 5 occurs.

7. Tally the terminated path {and note its coordinates, if you have time!}

8. Backtrack one level and repeat at Step 5.

Using this algorithm, suitably modified in implementation to cover both the square and the rectangular cases, and again, suitably modified for the evenness or oddness of N, yields the table of results, Table 1 attached.

N	2N	I,J {Square}	#	I,J {Rect}	#	C{2N}
2	4	1,0	2	1,0	1	5
		2,0	1	0,1	1	
3	6	1,0	12	1,0	4	37
		2,0	9	2,0	3	
		3,0	5	0,1	4	
4	8	1,0	212	1,0	43	782
		2,0	167	2,0	33	
		3,0	146	3,0	24	
		4,0	87	4,0	8	
				0,1	43	
				0,2	19	
5	10	1,0	11030	1,0	1026	44240
		2,0	8774	2,0	821	
		3,0	7538	3,0	670	
		4,0	7229	4,0	550	
		5,0	4522	5,0	243	
				0,1	1026	
				0,2	811	
6	12	1,0	<1.8x10^6>	⋮		<7.75x10^6>
		⋮		0,1	<1.25x10^5>	
		6,0	<6x10^5>	⋮		
				0,3	<5x10^4>	

<estimated>