Name : Shrinivas Hatyalikar
Div: TY-B (B2)
Roll No: 26

# Q) Tiling using L shaped trominos

**Approach:**

- Use recursion to divide the grid into quadrants.
- Depending on the position of the missing tile within a quadrant, place three tiles in specific positions to cover the missing tile.
- Continue recursively dividing and placing tiles until reaching a base case (2x2 grid), where tiles are placed to fill the quadrant.

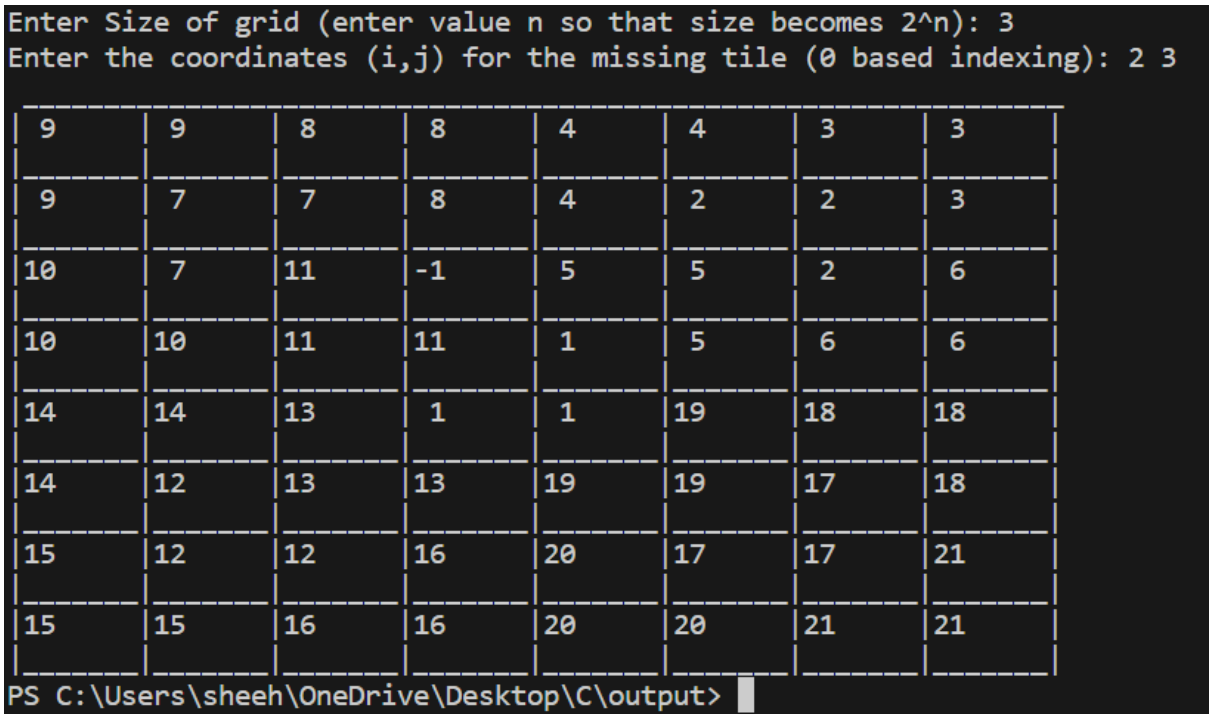
**Code:**

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

int size_of_grid, b, a, cnt = 0;
int arr[128][128];

void place(int x1, int y1, int x2,
           int y2, int x3, int y3)
{
    cnt++;
    arr[x1][y1] = cnt;
    arr[x2][y2] = cnt;
    arr[x3][y3] = cnt;
}

int tile(int n, int x, int y)
{
    int r, c;
    if (n == 2) {
        cnt++;
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (arr[x + i][y + j] == 0) {
                    arr[x + i][y + j] = cnt;
                }
            }
        }
        return 0;
    }
```

```
    for (int i = x; i < x + n; i++) {
        for (int j = y; j < y + n; j++) {
            if (arr[i][j] != 0)
                r = i, c = j;
        }
    }

    // If missing tile is 1st quadrant
    if (r < x + n / 2 && c < y + n / 2)
        place(x + n / 2, y + (n / 2) - 1, x + n / 2,
              y + n / 2, x + n / 2 - 1, y + n / 2);

    // If missing Tile is in 3rd quadrant
    else if (r >= x + n / 2 && c < y + n / 2)
        place(x + (n / 2) - 1, y + (n / 2), x + (n / 2),
              y + n / 2, x + (n / 2) - 1, y + (n / 2) - 1);

    // If missing Tile is in 2nd quadrant
    else if (r < x + n / 2 && c >= y + n / 2)
        place(x + n / 2, y + (n / 2) - 1, x + n / 2,
              y + n / 2, x + n / 2 - 1, y + n / 2 - 1);

    // If missing Tile is in 4th quadrant
    else if (r >= x + n / 2 && c >= y + n / 2)
        place(x + (n / 2) - 1, y + (n / 2), x + (n / 2),
              y + (n / 2) - 1, x + (n / 2) - 1,
              y + (n / 2) - 1);

    // dividing it again in 4 quadrants
    tile(n / 2, x, y + n / 2);
    tile(n / 2, x, y);
    tile(n / 2, x + n / 2, y);
    tile(n / 2, x + n / 2, y + n / 2);

    return 0;
}

void print_line(char ch, int times)
{
    for (int i = 0; i < times; i++)
        printf("%c", ch);
}

int main()
{
    printf("Enter Size of grid (enter value n so that size becomes 2^n): ");
    scanf("%d", &size_of_grid);
    size_of_grid = (int)pow(2, size_of_grid);
```

```
    // Initialize the array and set missing tile
    for(int i=0;i<size_of_grid;i++){
        for(int j=0;j<size_of_grid;j++){
            arr[i][j]=0;
        }
    }
    printf("Enter the coordinates (i,j) for the missing tile (0 based
indexing): ");
    scanf("%d %d", &a, &b);
    arr[a][b] = -1;

    tile(size_of_grid, 0, 0);

    char ab[] = "_______|";

    print_line(' ', 1);
    for (int j = 0; j < size_of_grid; j++)
    {
        print_line('_', 8);
    }
    printf("\n");

    for (int i = 0; i < size_of_grid; i++)
    {
        printf("|");
        for (int j = 0; j < size_of_grid; j++)
        {
            printf("%2d\t|", arr[i][j]);
        }
        printf("\n|");

        for (int j = 0; j < size_of_grid; j++)
        {
            printf("%s", ab);
        }
        printf("\n");
    }

    return 0;
}
```

## Output:


Enter Size of grid (enter value n so that size becomes 2^n): 3
Enter the coordinates (i,j) for the missing tile (0 based indexing): 2 3
| 9 | 9 | 8 | 8 | 4 | 4 | 3 | 3 |
| 9 | 7 | 7 | 8 | 4 | 2 | 2 | 3 |
|10 | 7 |11 |-1 | 5 | 5 | 2 | 6 |
|10 |10 |11 |11 | 1 | 5 | 6 | 6 |
|14 |14 |13 | 1 | 1 |19 |18 |18 |
|14 |12 |13 |13 |19 |19 |17 |18 |
|15 |12 |12 |16 |20 |17 |17 |21 |
|15 |15 |16 |16 |20 |20 |21 |21 |
PS C:\Users\sheeh\OneDrive\Desktop\C\output>