Pseudo code of problem 24 A

```
// Define constants and global variables
MAX_N = 64 // maximum value of N (the number of sides in the golygon)
MAX_M = 1024 // maximum number of blocked cells
MAX_WAYS = 1024 // maximum number of valid golygons
MAX_GRID = 4096 // maximum size of the grid
BASE = 1024 // a base value used to shift coordinates to positive integers

n, m, golygons = 0 // variables to store input and output values
g[MAX_GRID][MAX_GRID] = {}, g2[MAX_GRID][MAX_GRID] = {} // arrays to represent the grid and a temporary grid used during the DFS
path[MAX_N] = {} // array to store the current path being explored
x[MAX_M], y[MAX_M] = {} // arrays to store the coordinates of the blocked cells
sdir[] = "nsew" // array to map direction indices to direction letters
dx[] = {0, 0, 1, -1} // arrays to represent the four possible directions of movement
dy[] = {1, -1, 0, 0}
ways_count = 0 // variable to store the number of valid golygons found
ways[MAX_WAYS][MAX_N] = {} // array to store the valid golygons

// Define helper functions
make_pair(x, y) -> p // function to create a pair of integers
    p = {x, y}
    return p

pair_cmp(p1, p2) -> integer // function to compare two pairs of integers
    if p1.x != p2.x:
        return p1.x - p2.x
    else:
        return p1.y - p2.y

pair_equal(p1, p2) -> boolean // function to check if two pairs of integers are equal
    return (p1.x == p2.x) and (p1.y == p2.y)

set_init(s) -> None // function to initialize a set
    s.size = 0

set_insert(s, p) -> None // function to insert a pair of integers into a set
    s.arr[s.size] = p
    s.size = s.size + 1

set_find(s, p) -> boolean // function to check if a pair of integers is in a set
    for i in 1 to s.size:
        if pair_equal(s.arr[i], p):
            return True
    return False
```

```
set_clear(s) -> None // function to clear a set
    s.size = 0

dfs(x, y, dir, step) -> None // function to recursively explore all possible paths
    // Check if the current path is already too long to form a valid golygon
    if abs(x - 0) + abs(y - 0) > (step + n) * (n - step + 1) / 2:
        return

    // Check if the current path forms a complete golygon
    if step == n + 1:
        if x == 0 and y == 0:
            // Store the current path as a valid golygon
            path[step - 1] = '\0'
            strcpy(ways[ways_count], path)
            ways_count = ways_count + 1
            golygons = golygons + 1
        return

    // Try moving in each possible direction
    if dir != 0:
        // Try moving north or south
        for i in 0 to 1:
            tx = x
            ty = y
            ok = 1
            // Check if the new path overlaps with any blocked cells
            for j in 0 to step - 1:
                tx = tx + dx[i]
                ty = ty + dy[i]
                if g[BASE + tx][BASE + ty]:
                    ok = 0
                    break

            // If the new path is valid, recursively explore it
            if ok and not set_find(ban, make_pair(tx, ty)) and not g2[BASE + tx][BASE + ty]:
                g2[BASE + tx][BASE + ty] = 1
                path[step - 1] = sdir[i]
                dfs(tx, ty, 0, step + 1)
                g2[BASE + tx][BASE + ty] = 0

    if dir != 1:
        // Try moving east or west
        for i in 2 to 3:
            tx = x
            ty = y
            ok = 1
            // Check if the new path overlaps with any blocked cells
            for j in 0 to step - 1:
```

```
    tx = tx + dx[i]
    ty = ty + dy[i]
    if g[BASE + tx][BASE + ty]:
       ok = 0
        break

// If the new path is valid, recursively explore it
if ok and not set_find(ban, make_pair(tx,ty)) and not g2[BASE + tx][BASE + ty]:
    g2[BASE + tx][BASE + ty] = 1
    path[step - 1] = sdir[i]
    dfs(tx, ty, 1, step + 1)
    g2[BASE + tx][BASE + ty] = 0
```