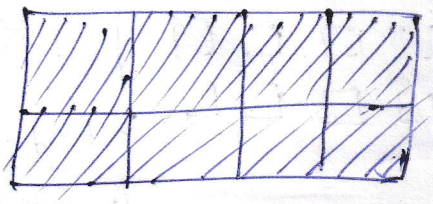
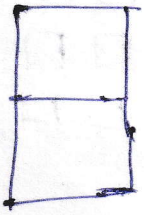
(1) Tiling Problem

Given a "2 x n" board and tiles of size "2 x 1", count the number of ways to tile the given board using these tiles.



2 x 4



2 x 1

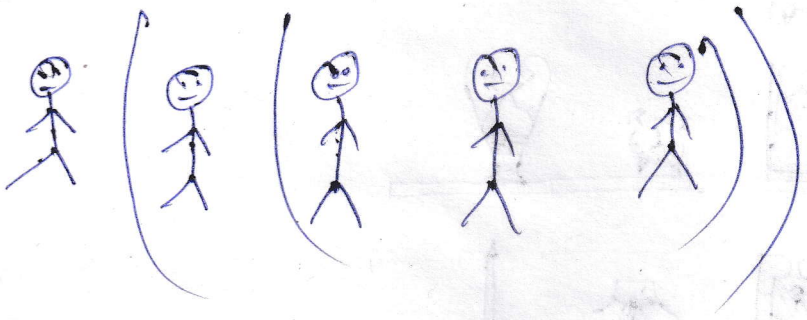
Code:

```
int tilingWays (int n) {
    if (n==0) {
        return 0;
    }
    if (n==1) {
        return 1;
    }
    return tilingWays (n-1) + tilingWays (n-2);
}
```

usage: tilingWays (4) => cout << ? << endl;

② Friends pairing problem

Find the number of ways in which n friends can remain single or can be paired up.



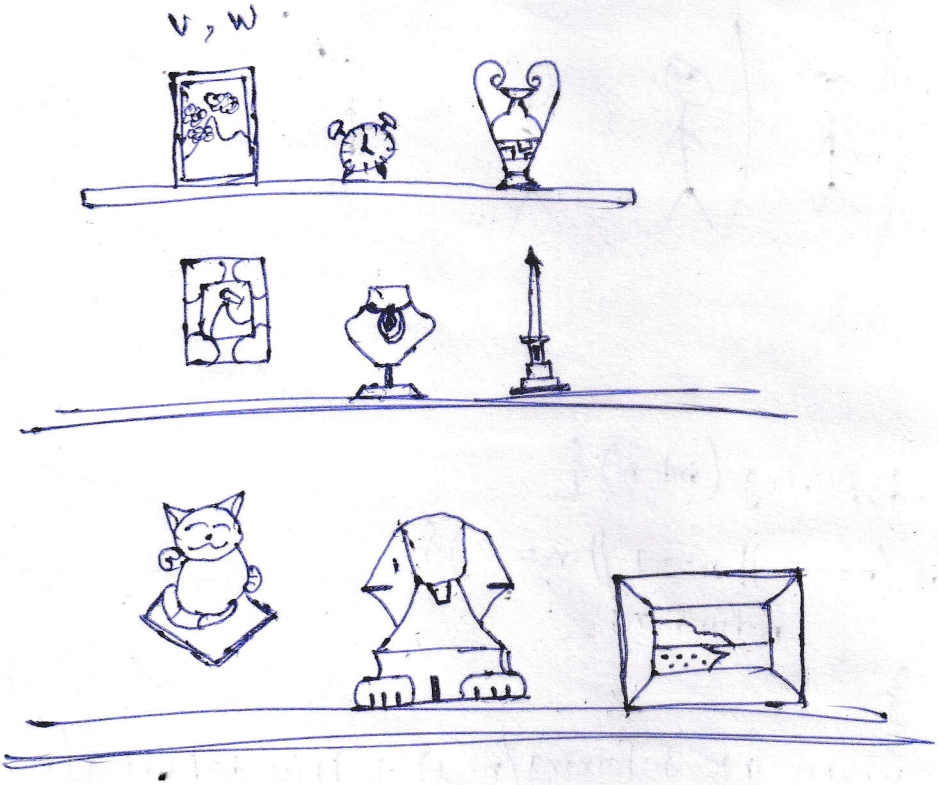
Code:

```
int friendsPairing (int n){
    if (n==0 || n==1 || n==2){
        return n;
    }
    return friendsPairing(n-1) + friendsPairing(n-2)*(n-1);
}
```

Usage: `cout << friendsPairing (int n) << endl;`

(3) 0-1 Knapsack Problem

Put n items with given weight and value in a Knapsack of capacity W to get the maximum total value in the knapsack.



i	0	1	2
wt [i]	10	20	30
value[i]	100	50	150

W = 50

knapsack.cpp

Code:

```
int knapsack (int value[], int wt[], int n, int W){
    if (n==0 || W==0){
        return 0;
    }
    if (wt[n-1] >W){
        return knapsack (value, wt, n-1, W);
    }

    return max(knapsack (value, wt, n-1, W - wt[n-1])+value[n-1],
               knapsack(value, wt, n-1, W));
}
```

Usage:

```
int wt[] = {10, 20, 30};
int value[] = {100, 50, 150};
int W = 50;

cout << knapsack(value, wt, 3, W) << endl;
```