Egg drop C++

```cpp
#include <iostream>
#include <climits>
using namespace std;

int eggDrop(int n, int k) {
   // Initialize a 2D array for DP table
   int dp[n + 1][k + 1] = {0}; // Array with (n + 1) rows and (k + 1) columns

   // Fill the DP table
   for (int i = 1; i <= n; i++) {
      for (int j = 1; j <= k; j++) {
         if (i == 1) {
            dp[i][j] = j; // If only one egg is available, we need j trials
         } else if (j == 1) {
            dp[i][j] = 1; // If only one floor is there, one trial needed
         } else {
            int minDrops = INT_MAX;
            // Check all floors from 1 to j to find the minimum drops needed
            for (int floor = 1; floor <= j; floor++) {
               int breaks = dp[i - 1][floor - 1]; // Egg breaks, check below floors
               int survives = dp[i][j - floor]; // Egg survives, check above floors
               int maxDrops = 1 + max(breaks, survives); // Maximum drops needed in worst case
               minDrops = min(minDrops, maxDrops); // Minimum drops to find the critical floor
            }
            dp[i][j] = minDrops;
         }
      }
   }

   return dp[n][k]; // Return the minimum drops needed
}

int main() {
   int n = 4; // Number of eggs
   int k = 2; // Number of floors

   cout << eggDrop(n, k) << endl; // Output the minimum drops required
   return 0;
}
```

Dry Run of the Program

Let's go through the dry run of the eggDrop function with n = 4 (number of eggs) and k = 2 (number of floors).

Input:

- n = 4 (number of eggs)
- k = 2 (number of floors)

Step 1: Initialize the DP Table

The DP table is a 2D array of size (n+1) x (k+1):

int dp[n+1][k+1] = {0}; // Array with 5 rows (0 to 4 eggs) and 3 columns (0 to 2 floors)

So, the DP table initially looks like this:

```
dp[0] = {0, 0, 0}  // 0 eggs: impossible to drop
dp[1] = {0, 0, 0}  // 1 egg
dp[2] = {0, 0, 0}  // 2 eggs
dp[3] = {0, 0, 0}  // 3 eggs
dp[4] = {0, 0, 0}  // 4 eggs
```

Step 2: Fill the DP Table

- **Case 1: One egg (i = 1)**

If we have only one egg (i = 1), the number of trials needed to check all j floors is equal to j because we must start from the 1st floor and test each floor one by one. This is why dp[1][j] = j.

So, for i = 1:

- dp[1][1] = 1 (1 trial for 1 floor)
- dp[1][2] = 2 (2 trials for 2 floors)

At this point, the table looks like this:

```
dp[0] = {0, 0, 0}
dp[1] = {0, 1, 2}
dp[2] = {0, 0, 0}
dp[3] = {0, 0, 0}
dp[4] = {0, 0, 0}
```

- **Case 2: One floor (j = 1)**

If we have only one floor (j = 1), then only 1 trial is needed, no matter how many eggs we have. So, for all i (eggs), dp[i][1] = 1.

At this point, the table becomes:

```
dp[0] = {0, 0, 0}
dp[1] = {0, 1, 2}
```

	dp[2] = {0, 1, 0}
	dp[3] = {0, 1, 0}
	dp[4] = {0, 1, 0}
	
	• **Case 3: More than 1 egg and more than 1 floor (i > 1, j > 1)**
	
	Now, we compute the minimum number of trials for each i (eggs) and j (floors) by testing each floor from 1 to j and determining the worst-case number of drops.
	
	For each floor floor:
	
	• If the egg breaks, we look at the number of drops for i - 1 eggs and floor - 1 floors (dp[i - 1][floor - 1]).
	• If the egg survives, we look at the number of drops for i eggs and j - floor floors (dp[i][j - floor]).
	• The worst-case number of drops is 1 + max(breaks, survives).
	• We want to minimize this worst-case number of drops.
	
	Let's calculate the values for each combination of i and j.
	
	For i = 2, j = 2:
	
	• Try floor 1:
	  ○ If the egg breaks, check dp[1][0] (0 floors) → 0 drops.
	  ○ If the egg survives, check dp[2][1] (1 floor) → 1 drop.
	  ○ So, max(0, 1) + 1 = 2 drops.
	
	Therefore, dp[2][2] = 2.
	
	The table becomes:
	
	dp[0] = {0, 0, 0}
	dp[1] = {0, 1, 2}
	dp[2] = {0, 1, 2}
	dp[3] = {0, 1, 0}
	dp[4] = {0, 1, 0}
	
	For i = 3, j = 2:
	
	• Try floor 1:
	  ○ If the egg breaks, check dp[2][0] → 0 drops.
	  ○ If the egg survives, check dp[3][1] → 1 drop.
	  ○ max(0, 1) + 1 = 2 drops.
	
	Therefore, dp[3][2] = 2.

	The table becomes:
	
	dp[0] = {0, 0, 0}
	dp[1] = {0, 1, 2}
	dp[2] = {0, 1, 2}
	dp[3] = {0, 1, 2}
	dp[4] = {0, 1, 0}
	
	For i = 4, j = 2:
	
	• Try floor 1:
	  ○ If the egg breaks, check dp[3][0] → 0 drops.
	  ○ If the egg survives, check dp[4][1] → 1 drop.
	  ○ max(0, 1) + 1 = 2 drops.
	
	Therefore, dp[4][2] = 2.
	
	The table becomes:
	
	dp[0] = {0, 0, 0}
	dp[1] = {0, 1, 2}
	dp[2] = {0, 1, 2}
	dp[3] = {0, 1, 2}
	dp[4] = {0, 1, 2}
	
	Step 3: Output the Result
	
	The final value in dp[n][k] (i.e., dp[4][2]) is 2.
	
	So, the minimum number of trials needed to find the critical floor with 4 eggs and 2 floors is 2.

Output:- 2	